

# **Discrete Abstractions of Hybrid Systems: Verification of Safety and Application to User-Interface Design**

*Meeko Oishi, Claire Tomlin, and Asaf Degani*

## The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the Lead Center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

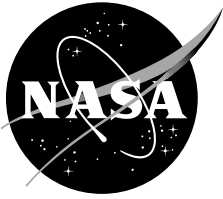
- **TECHNICAL PUBLICATION.** Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.
- **TECHNICAL MEMORANDUM.** Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- **CONTRACTOR REPORT.** Scientific and technical findings by NASA-sponsored contractors and grantees.

- **CONFERENCE PUBLICATION.** Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.
- **SPECIAL PUBLICATION.** Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- **TECHNICAL TRANSLATION.** English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at <http://www.sti.nasa.gov>
- E-mail your question via the Internet to [help@sti.nasa.gov](mailto:help@sti.nasa.gov)
- Fax your question to the NASA Access Help Desk at (301) 621-0134
- Telephone the NASA Access Help Desk at (301) 621-0390
- Write to:  
NASA Access Help Desk  
NASA Center for AeroSpace Information  
7121 Standard Drive  
Hanover, MD 21076-1320



## **Discrete Abstractions of Hybrid Systems: Verification of Safety and Application to User-Interface Design**

*Meeko Oishi*

*Stanford University, Stanford, California*

*Claire Tomlin*

*Stanford University, Stanford, California*

*Asaf Degani*

*Ames Research Center, Moffett Field, California*

National Aeronautics and  
Space Administration

Ames Research Center  
Moffett Field, California 94035-1000

Available from:

NASA Center for AeroSpace Information  
7121 Standard Drive  
Hanover, MD 21076-1320  
(301) 621-0390

National Technical Information Service  
5285 Port Royal Road  
Springfield, VA 22161  
(703) 487-4650

## Abstract

Human interaction with a complex control system involves the user, the automation's discrete mode logic, and the underlying continuous dynamics of the physical system. The user-interface of such systems always displays a reduced set of information about the entire system. Designing interfaces such that all the pertinent information is available and assuring that this information is correct is important for any user-interface, but especially so for safety-critical systems such as automotive systems and autopilots. Here we describe a methodology for the analysis of hybrid control systems that incorporate user interaction, with the goal of assuring that the information provided to the user is correct. That is, the user-interface must contain all information necessary to safely complete a desired procedure or task.

We begin with a hybrid system model which incorporates discrete mode logic as well as nonlinear continuous dynamics. Using a hybrid computational tool for reachability, we find the largest region of the state-space in which we can guarantee the state of the system can always remain – this is the safe region of operation. By implementing a controller for safety which arises from this computation, we mathematically guarantee that this safe region is invariant, meaning that the system will always remain within the safe region if the determined controller is used on the boundary of the safe region. Verification within a hybrid framework allows us to account for the continuous dynamics underlying the discrete representations displayed to the user. Using the computed invariant regions as discrete states, we can abstract a discrete event system from this hybrid system with safety restrictions. This abstraction can be used to determine what information must be provided on the display. Furthermore, in cases in which an interface already exists, the abstraction provides the necessary input into existing interface verification methods.

We provide two examples: a car traveling through a yellow light at an intersection and an aircraft autopilot in an automatic landing/go-around maneuver. The examples demonstrate the applicability of this methodology to hybrid systems that have operational constraints we can pose in terms of *safety*. This methodology differs from existing work in hybrid system verification in that we directly account for the user's interactions with the system.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Problem Definition . . . . .	3
1.2	Approach . . . . .	4
1.3	Organization . . . . .	6
<b>2</b>	<b>The Yellow Interval Dilemma</b>	<b>7</b>
2.1	Implications for Interface Design . . . . .	11
<b>3</b>	<b>Methodology</b>	<b>16</b>
3.1	Mathematical Formulation of the Problem . . . . .	16
3.2	Step 1: Separation into Hybrid Subsystems . . . . .	18
3.3	Step 2: Hybrid Reachability Analysis and Controller Synthesis . . . . .	19
3.4	Step 3: Abstraction: Hybrid $\rightarrow$ Discrete . . . . .	24
3.5	Use of the Discrete Abstraction . . . . .	29
<b>4</b>	<b>Automatic Landing of a Commercial Aircraft</b>	<b>32</b>
4.1	Hybrid Procedural Automaton . . . . .	34
4.2	Reachability Analysis of Hybrid Subsystems . . . . .	37
4.3	Discrete Abstraction . . . . .	38
4.4	Implications for User-Interface Analysis . . . . .	39
<b>5</b>	<b>Implications for User-Interface Design and Analysis</b>	<b>42</b>
<b>6</b>	<b>Conclusions and Limitations</b>	<b>44</b>

# 1 Introduction

## 1.1 Problem Definition

Human interaction with automation is pervasive: almost every aspect of our lives involves computer systems, information systems, machines, and devices. Whether it is a household device (such as a microwave oven, an alarm clock, or a VCR), a heating/cooling system, a navigation system in a car, or an autopilot in modern aircraft, the human operator is provided with information about the automation and the processes it controls. With the exception of fully-automated systems, the operator is also provided with the means to supervise the process: changing modes and behaviors, or setting parameters, for example. Interaction with these systems is provided through an *interface*, which conveys information about the underlying system dynamics and processes, and includes a control panel through which the user can enter inputs (such as mode changes).

Complex automated systems are composed of many states, modes, events, parameters and protocols. The user's interactions with the automation are determined by the interface, always an abstracted description of the underlying behavior of the machine [1]. Too much information can overwhelm the user; not enough means the user may not be able to perform the desired task. In commercial aircraft, the FAA mandates extensive bench tests, simulator tests, and in-flight tests to certify aircraft autopilots [2]. However, it is difficult to anticipate all possible combinations of system states, modes, and environmental behaviors. There is an increasing need to analyze systems and interfaces before production or even before prototyping in safety-critical, expensive, or high-risk applications. While analysis of smaller systems can be completed ad hoc, analysis of larger and more complex systems requires systematic and rigorous methods.

Human-automation interaction in aviation has been a controversial topic since the advent of computers and their integration into the cockpit [3, 4, 5]. Since the introduction of the glass cockpit in the mid-1980s, pilots have voiced concern regarding their ability to fully understand and control the automation. Indeed, the aviation industry has experienced many incidents and some accidents in which the pilots became confused about the current mode or could not anticipate the next mode in the automation [6, 7, 8]. The problem has been loosely termed *mode confusion*. Although the engineering psychology community has historically dominated research on human-automation interaction, there have recently been efforts by the formal methods community [9, 10, 11, 12, 13] as well as system and control communities [14, 15, 16] to address these safety-critical problems.

To analyze and identify the type of problems particular to human interaction with complex systems, one first needs to describe and represent the underlying system. In aircraft, for example, behavior is controlled through both continuous and discrete elements: flight control systems continuously regulate aerodynamic surfaces, and autopilot systems determine the discrete mode-logic

of the aircraft (such as the application of altitude-holding modes or vertical speed-holding modes). Such systems, which couple continuous dynamics and control systems with a “higher level” discrete logic control, are known as *hybrid* systems. Whereas automated tools which verify properties for discrete logic systems have been used for some time, the development of such tools for hybrid systems is an area of current active research.

## 1.2 Approach

In this report, we use a recently developed methodology in hybrid system reachability and controller synthesis [17, 18, 19, 20] to analyze highly-automated, hybrid systems that are supervised by humans. Verification is defined simply as the process of developing and executing a mathematical proof that a system model satisfies a given specification. The ability to verify specifications about a system model allows for heightened confidence that the actual system will perform as intended. Methods and tools to verify systems have become paramount as the complexity of automated systems has grown; it is no longer possible to rely on intuition, testing, and simulation to assure that a system satisfies its specification. With a keen interest in cost-efficient design processes, most hardware and some software companies have embraced verification techniques as part of their product design process: verification is one of the key enabling technologies for increased automation. Verification tools can aid in drastically reducing time spent on design and validation, but are also crucial in ensuring that safety properties are upheld. In safety-critical applications such as airbag deployment circuitry, medical devices, and aircraft autopilots, guarantees of safe operation are paramount.

In November 1994, Intel’s infamous “floating-point-division” bug was discovered, eventually forcing recall of the Pentium chip and leading to approximately US \$500 million in charges against Intel [21]. The cause of the bug was a logical error, in which a look-up table to aid calculating quotients of floating point numbers was erroneously indexed [21, 22]. Identifying these errors before mass production is of the utmost importance, given the expensive and even potentially dangerous nature of these mistakes. However, chip-making companies are not the only ones with vested interests in verification. Widespread use of automation in complex, safety-critical systems such as aviation, has also led to an interest in the verification of *human-automation systems*, in which humans interact with highly automated systems.

Formal verification and design of interfaces focuses on the information content of the display, rather than the design of the graphical user interface [1, 23, 9, 24, 10, 11, 25]. It addresses the acute problem of detecting design errors in human-machine interaction and verifying the correctness of the interaction in complex and automated control systems. Recently, a theory, methodology, and a detailed verification procedure was developed by researchers at NASA [11, 25]. The methodology

considers four elements: the machine model, the user model, the interface, and the task specification. In [1], the authors address the automatic construction of simple and succinct interfaces for highly complex systems. The cornerstone of the approach is the recognition that an interface is an abstracted representation of the underlying system. Using this idea, discrete-state reduction techniques may be used to abstract away superfluous information and construct a minimal interface. To this end, algorithms for verification and construction have been developed and applied to the verification and design of automated flight control systems [11, 25]. In [11], the hybrid plant model is represented as a discrete system in which the system dynamics are modeled as plant-triggered (dynamic) transitions.

Hybrid reachability analysis and controller synthesis addresses the problem of guaranteeing system *safety*. Many problems of interest may be posed as reachability specifications on the system’s set of continuous states. For example, in the problem of verifying system safety, the safety specification is first represented as a desired subset of the state space in which the system should remain, and then the following reachability question is posed: *Do all trajectories of the system remain within this set of desired states?* The process of verifying safety then involves computing the subset of the state space which is backwards reachable from this “safe set” of states; if this backwards reachable set intersects any states outside the desired region, then the system is deemed unsafe. *Controller synthesis for safety* is a procedure through which one attempts, by restricting the behavior of the system through a controller, to prune away system trajectories which lead to unsafe states.

In the past several years, a method [17, 18] and a numerical tool [19, 20, 26] have been developed for verifying the safety of hybrid systems. The hybrid system model combines continuous state and discrete event models: the discrete components represent different modes of the system and the actions used to transition between modes, whereas the continuous components model the physical process itself, such as the continuous response of an aircraft to the forces due to control surfaces and engine thrust, and evolve according to smooth mode-dependent dynamics which incorporate continuous inputs.

Applications of hybrid system theory to automated systems have traditionally assumed that the controller itself is an automaton which runs in parallel with the system under control. Here we consider the problem of controlling *human-automation* systems, in which the automaton and a human operator share authority over the control of the system. In particular, we consider the problem of analyzing user-interaction with hybrid systems. The main contribution of this report is to show how hybrid human-automation systems, with the aid of the hybrid system reachability tool, can be transformed into an equivalent discrete representation, which could later be incorporated into the design of user-interfaces.

### 1.3 Organization

We first consider the problem of creating a discrete abstraction of a hybrid human-automation system within the framework of an everyday driving problem: braking before or driving through an intersection at a yellow light. Using this example as a backdrop, we describe a methodology for verification of hybrid systems that incorporate user-interaction. The methodology has three components: separation of the original hybrid system into subsystems, each of which contains no user-controlled transitions between modes; reachability analysis and controller synthesis; and abstraction to a discrete system. Each component is illustrated through the driving example. Implications for interface design are discussed in detail. The second example concerns the automatic landing function of a highly automated, safety-critical, commercial et aircraft.

These two examples demonstrate the types of problems this verification and abstraction methodology can address. The hybrid and discrete modeling framework and analysis used in this method makes certain assumptions about relevant engineering, operational, and human-factors issues. We discuss these limitations, as well as possible directions for future work in Section 6.

## 2 The Yellow Interval Dilemma

Consider the following scenario: A single driver on an expressway approaches an intersection when suddenly the light turns yellow. The driver must decide whether to brake and try to stop at the intersection, or to continue driving through the intersection before the red light appears. Typically, the driver has an “intuitive” feel for the correct course of action [27], due to accumulated experience about the particular car’s braking capabilities, the duration of the yellow light at a given intersection, and the road and weather conditions. However, many drivers might have had first-hand experience to the contrary: these drivers slow down, only to realize (too late) that the car cannot stop before the intersection, and then attempt to accelerate through the intersection.

There are several factors influencing the correct course of action at a yellow light. (By ‘correct’, we mean a course of action that will allow the driver to avoid a *red light violation*, which occurs whenever the car is in the intersection at any time during a red light.) Physical constraints include not only the car’s limitations (i.e. braking and acceleration capabilities) but also the duration of the yellow light (known as the *yellow interval*) as well as the posted speed limit. We can plot these constraints for reasonable distances  $x$  from the intersection and reasonable speeds  $v$  at which the car can travel. In Figure 1, notice that along the horizontal axis,  $x = 0$  denotes the side of the intersection closest to the approaching car, and  $x = L$  denotes the far side of the intersection, where  $L = 10$  m is the length of the intersection. Along the vertical axis, we examine positive speeds  $v$ , up to the speed limit  $v_{\max} = 24$  m/s (approximately 50 mph)<sup>1</sup>. The car has a range of braking and acceleration capabilities: it is not possible to brake more than the maximum braking or to accelerate more than the maximum acceleration. The yellow interval is a fixed duration  $\tau$ , and we assume that the car remains at or below the posted speed limit  $v_{\max}$ . Another important factor to consider is the driver’s reaction time: when the light switches from green to yellow, there is a finite amount of time  $\Delta$  before the driver can react to the change. While the driver’s reaction time varies per individual and per circumstance, for speeds considered in this example, a reasonable reaction time for an average, alert driver, is  $\Delta = 1.5$  seconds [27, 28, 29]. Taking these factors into consideration, the question we wish to answer is: What actions does the driver need to take, and when does the driver need to take them, to avoid being in the intersection during a red light?

We first determine where the car can be, relative to the intersection, so that it can stop at the intersection during the yellow or red light. If the car stops right at the intersection, and we know the driver used maximum braking to reach this point, we can determine where the driver could

---

<sup>1</sup>For the analysis that follows, we assume that during the green light, the car must travel at a positive speed  $v > 0$  (the car is not allowed to come to a complete stop anywhere along the expressway); and during the yellow and red lights, the car must travel at a positive speed  $v > 0$  for all  $x$  except at the intersection  $x = 0$  (the car is only allowed to come to a complete stop exactly at the intersection).

have been  $\tau + \tau_{\text{red}}$  seconds before, where  $\tau_{\text{red}}$  is the duration of the red light<sup>2</sup>. Modeling the car as a point-mass, the car's distance from the intersection is denoted  $x$ , the car's speed is denoted  $v = \dot{x}$ , and the control input  $u$  (which is the deceleration due to braking) is the second derivative of position  $u = \ddot{x}$ . We integrate this system backwards in time with  $u = u_{\text{min}}$  (for the maximum braking force) from  $x(0) = 0, v(0) = 0$ , which represents the car at a full stop at the intersection. For a typical passenger car,  $u_{\text{min}} = -4 \text{ m/s}^2$ , and for a typical intersection, the duration of the yellow light is  $\tau = 4$  seconds [28, 30] and the duration of the red light is  $\tau_{\text{red}} = 20$  seconds. To account for the driver's reaction time, we assume that when the yellow light first appears, the car continues at its current speed  $v$  for  $\Delta$  seconds before the driver can take any action. Therefore we only integrate the system for  $\tau + \tau_{\text{red}} - \Delta$  seconds. The result of this analysis shows that there are certain combinations of speed and distance from the intersection from which the driver will simply not be able to stop before reaching the intersection. (The details of this analysis will be presented in Section 3.3.)

Now we analyze the second option: driving through the intersection. We wish to know where the car can be, relative to the intersection, in order that the driver may drive through the intersection before the light turns red. While driver manuals and automobile insurers certainly do not recommend it as a practice, many drivers accelerate when a yellow light occurs. If the driver accelerates with  $u = u_{\text{max}}$ , where for a typical passenger car the maximum acceleration is  $u_{\text{max}} = 2 \text{ m/s}^2$ , we can compute how far from the intersection the car must be when the yellow light first appears, in order to avoid a red light violation. In this calculation, we assume the driver stops accelerating and maintains  $v = v_{\text{max}}$  once the car reaches the speed limit. If, alternatively, the driver simply coasts through the intersection with a constant speed,  $u = 0$ , we can compute how far from the intersection the car must be in order to coast completely through the intersection. Again, assuming that in the worst-case scenario the car just reaches the far side of the intersection as the light turns red, we determine where the driver could have been  $\tau$  seconds previous to this by examining first where the driver could have been  $\tau - \Delta$  seconds previously (through acceleration with  $u = u_{\text{max}}$  or through coasting with  $u = 0$ ), then where the driver could have been  $\Delta$  seconds previous to that (through coasting with  $u = 0$ ). Mathematically, for an intersection of length  $L$ , we first integrate  $\ddot{x} = u$  with  $u = u_{\text{max}}$  (for the case in which the car accelerates) or  $u = 0$  (for the case in which the car travels at a constant speed) from  $x(0) = L, v(0) \in [0, v_{\text{max}}]$  backwards in time for  $\tau - \Delta$  seconds. We then integrate  $\ddot{x} = u$  with  $u = 0$  from this result backwards in time for  $\Delta$

---

<sup>2</sup>In some cities, the front of the car (denoted by a distance  $d_{\text{front}}$ , measured from the front bumper to the mid-point of the front wheel) is actually allowed to be in the intersection during the red light. While our model treats the car as a point-mass, to account for this rule in our analysis, we could instead examine where the car could be relative to  $x = d_{\text{front}}$ , and integrate backwards in time for  $\tau + \tau_{\text{red}}$  from this point instead of  $x = 0$ . The method proceeds similarly as presented here. Alternatively, in another variation, the car can be completely inside the intersection when a red light occurs without incurring a red light violation. In our analysis, for a car of length  $d_{\text{car}}$ , we would first integrate from  $x = d_{\text{car}}$  instead of  $x = L$ . The rest of the analysis proceeds similarly as presented above.

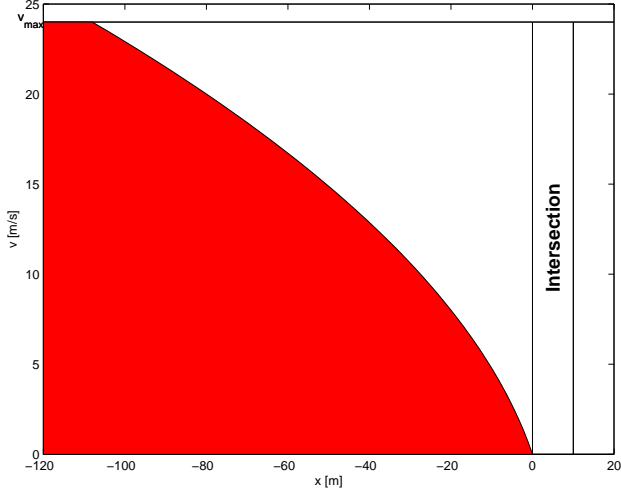


Figure 1: The shaded (red) region shows the combination of  $x$  and  $v$  for which the car can stop at the intersection during a yellow or red light. Along the curved section of the boundary of the shaded region, the driver must use  $u = u_{\min}$  in order to stop at the intersection.

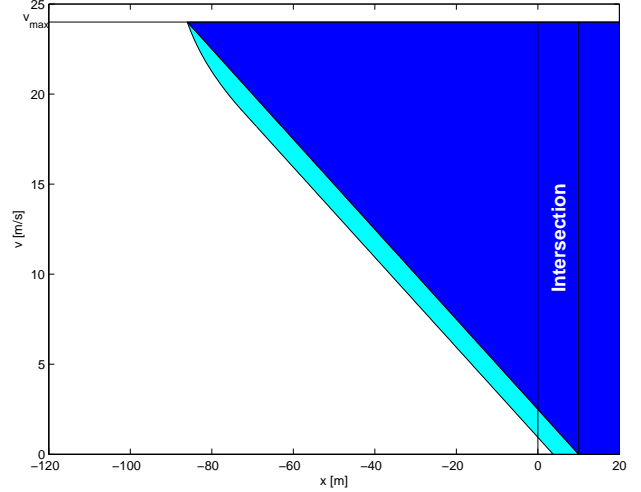


Figure 2: The darker shaded (dark blue) region shows the region of the state-space from which coasting ( $u = 0$ ) during the yellow interval will allow the car to completely cross the intersection by the time the red light appears. The lighter shaded (cyan) region shows the region of the state-space from which acceleration at  $u = u_{\max}$  (until the speed limit is reached) will allow the car to completely cross the intersection before the red light appears.

seconds. The analysis shows that at lower speeds, the driver must be closer to the intersection to be able to drive through the intersection before the red light appears.

We can depict these two scenarios graphically in the continuous state-space, for reasonable distances  $x$  and speeds  $v$ . Figure 1 shows the maximum braking scenario: the shaded (red) region depicts the combinations of distance and speed for which the driver will be able to stop at the intersection. If the driver attempts to brake in the unshaded (white) region, even braking as hard as possible, the car will end up in the middle of the intersection. Additionally, when the car reaches the boundary of the shaded (red) region, the driver *must* enforce maximum braking  $u = u_{\min}$ : any other choice of  $u$  (anything other than maximum braking) will inevitably result in a red light violation.

Similarly, Figure 2 shows the acceleration/coasting scenario: the darker shaded (dark blue) region depicts the combinations of distance and speed for which the driver can simply coast through the intersection and reach the end of the intersection just as the light turns red. If the driver attempts to coast through when the car is outside of the darker shaded (dark blue) region, the car will still be caught in the intersection during the red light. Alternatively, the lighter shaded (cyan) region depicts the combinations of distance and speed for which the driver must accelerate

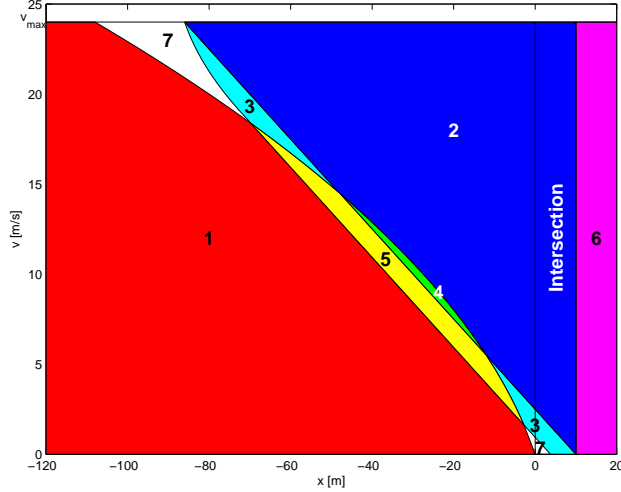


Figure 3: During the yellow light, the following state space can be divided into seven distinct regions with respect to the car’s capabilities: (1) Safe to stop at the intersection, (2) Safe to coast or accelerate through the intersection, (3) Safe to accelerate through the intersection, (4) Safe to coast or accelerate through, or to stop at the intersection, (5) Safe to accelerate through, or to stop at the intersection, (6) Safe (already through the intersection), and (7) Not safe to stop at the intersection and not safe to drive through the intersection.

with  $u = u_{\max}$  in order to avoid a red light violation. (The driver could also accelerate in the darker shaded (dark blue) region, but does not have to.) If the yellow light appears when the car is outside of both of the shaded (dark blue and cyan) regions, the car will not be able to cross the intersection before the red light appears. In the unshaded (white) region, there is no control law (given  $u \in [u_{\min}, u_{\max}]$ ) which will allow the car to avoid a red light violation.

To consider the possible driver actions during the yellow interval, we combine these two results (from Figures 1 and 2) in Figure 3. We identify seven distinct regions: (1), (2), (3), (4), (5), (6), and (7). There are some combinations of speed and distance for which the driver must eventually brake in order to avoid a red light violation (1), some for which the driver must coast or accelerate (2), some for which the driver must accelerate (3), some for which the driver can either brake or accelerate (4), some for which the driver can either brake, coast, or accelerate (5), some for which the driver can drive unconstrained (6), and, most troubling, some for which the driver cannot do any of the above (7). When the car is in region (7) and a yellow light appears, the driver will inevitably end up in the intersection during the red light! The driver is too close to the intersection to be able to stop before the intersection, and too far away to coast safely through it.

This problem is known as the “yellow interval dilemma” [30, 31]. The portion of region (7) at high speeds (near the speed limit) is called the “dilemma zone”. A common and well-publicized problem [32] for traffic engineers is to design the yellow interval  $\tau$  to minimize red light violations. To minimize (or hopefully eliminate) the dilemma zone, the Institute of Traffic Engineers recommends

a variety of heuristic additions to the kinematic formula for braking. These additions are based on a typical speed for most cars at a particular intersection, the length of the intersection, and the presence of a “red light interval” (in which all directions at an intersection display red at the same time) [28]. The dilemma zone can be eliminated by increasing the duration of the yellow interval  $\tau$ , or alternatively by decreasing the speed limit  $v_{\max}$ . (Increasing  $\tau$  decreases the slope of the boundary of the region for safe coasting (region (2) and region (4)).) Additionally, we must consider the human factor: increasing  $\tau$  beyond approximately 5 seconds has been shown experimentally to actually increase the number of red light violations at a given intersection: drivers quickly learn that the yellow light is extremely long, and tend not to brake when encountering the long yellow light [29].

To eliminate any possibility of a red light violation, the driver must never enter the unsafe region (7) during a green light. If the car is never allowed to be in (7) during a green light, the car will never be in the dilemma zone when the yellow light occurs. We again propagate the dynamics  $\ddot{x} = u$ ,  $u \in [u_{\min}, u_{\max}]$ , from the boundaries of the unsafe region (7) to determine for which combinations of  $x$  and  $v$  there is a control  $u$  which will keep the car from entering the unsafe region (7). We additionally constrain the dynamics so that  $v > 0$  (we do not allow the vehicle to come to a complete stop).

The result is shown in Figure 4. During the green light, the shaded (cyan) area is “safe” to operate in: this is the region for which we know the car can always avoid a red light violation, if a yellow light suddenly appeared. Note that the unshaded (white) region is slightly larger than the region (7) of Figure 3 at low speeds: since the car cannot come to a complete stop during the green light, the car must accelerate to “miss” the unsafe region. However, the region which accounts for the dilemma zone is unchanged. Along the boundaries shown, certain control laws must be enforced (the car must decelerate with  $u = u_{\min}$  along the upper left boundary of the dilemma zone, for example).

This analysis has answered the original question: it has shown exactly what actions the driver needs to enact, as well as when to enact them, in order to prevent a red light violation. The driver must follow different control actions depending on the car’s current position and speed, as well as the color of the light: the restrictions on these actions differ during green, yellow, and red traffic lights.

## 2.1 Implications for Interface Design

To provide this information to the driver, an add-on dashboard device with sensor access to distance from the intersection (Figure 9) could be used. This device could use the results of the previous

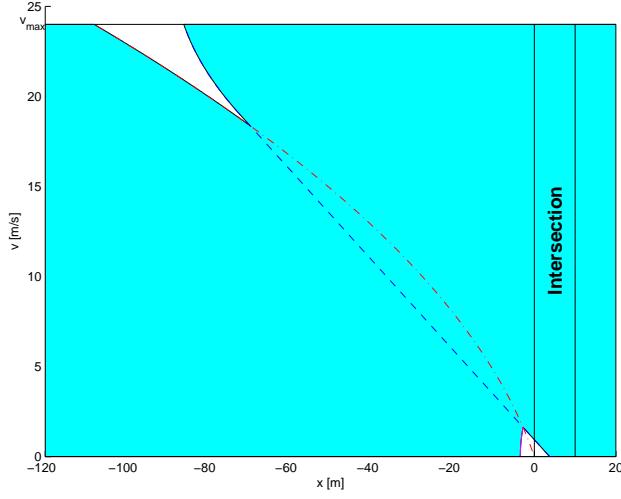


Figure 4: Safe region of operation (shaded, cyan) during the green light to avoid being caught in the dilemma zone during the yellow interval. The acceleration and braking curves are shown in dashed and dashed-dotted lines, respectively, for clarity.

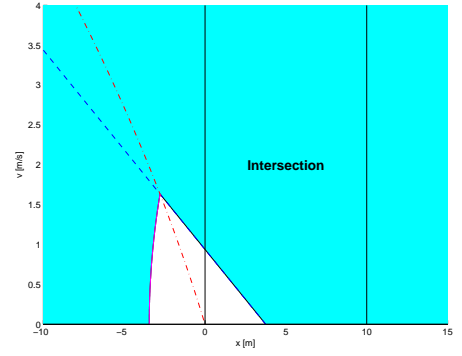


Figure 5: Close view from Figure 4 of the increased unsafe region (unshaded, white) at low speeds to the left of the intersection. This new boundary is obtained by integrating the point at the intersection of the braking curve (dashed-dotted line) and the acceleration curve (dashed line) backwards in time. The unsafe region corresponding to the dilemma zone is unaffected.

Light Color	Region	Interface Indication
Green (Figure 4)	shaded	CONTINUE DRIVING
	top boundary	BRAKE
	bottom boundary	ACCELERATE
	unshaded	UNSAFE
Yellow (Figure 3)	(1), (4), (5)	CONTINUE DRIVING
	boundary of (1) and (4)	BRAKE
	(3)	ACCELERATE
	(2)	ACCELERATE OR MAINTAIN CURRENT SPEED
	(6)	CONTINUE DRIVING
	(7)	UNSAFE
Red (Figure 3)	(1), (4), (5)	CONTINUE DRIVING
	boundary of (1) and (4)	BRAKE
	(6)	CONTINUE DRIVING
	(2), (3), (7)	UNSAFE

Table 1: Summary of abstraction from continuous state-space regions to interface indications during green, yellow, and red lights. These abstractions, determined by control law restrictions necessary to avoid a red light violation, are shown graphically in Figures 6, 7, and 8, respectively.

analysis (Figures 3 and 4) to advise the driver. In order to keep the advice simple, the analysis allows entire regions of the  $(x, v)$ -space to be abstracted into simple, discrete indications.

For example, consider such a device which has four advisories: “CONTINUE DRIVING”, “BRAKE”, “ACCELERATE”, and “ACCELERATE OR MAINTAIN CURRENT SPEED”. During the green light, we can abstract all of the shaded region of Figure 4 to the single discrete representation “CONTINUE DRIVING”. This indicates that the driver is free to use any control  $u \in [u_{\min}, u_{\max}]$ , accelerating and decelerating at will. However, along the boundary, certain controls must be enforced, as shown in Figure 6: along the dashed-bolded boundary,  $u = u_{\min}$ , and along the solid-bolded boundary,  $u = u_{\max}$ . We can abstract each of these boundaries into simple representations: “BRAKE” and “ACCELERATE”, respectively. These abstractions, as well as those for the regions relevant during the yellow and red intervals, are summarized in Table 1. Note that during the yellow interval, we assume braking to stop before the light is the preferred course of action when it is safely possible. Figures 6, 7, and 8 show the final abstractions as well as the control laws which dictate the abstraction, during the green, yellow, and red traffic lights, respectively.

In this example, the interface indications are inspired by analysis of the system in terms of its *safety*, in which the car is “safe” if it can avoid a red light violation. Although there are an infinite number of conditions to consider, this analysis allows us to abstract these conditions into a finite set of simple indications which represent, according to our model, everything that the driver must know in order to safely interact with the system. The proposed interface (Figure 10) tells the driver (based on the car’s speed and distance to the intersection) whether to stop at the intersection or

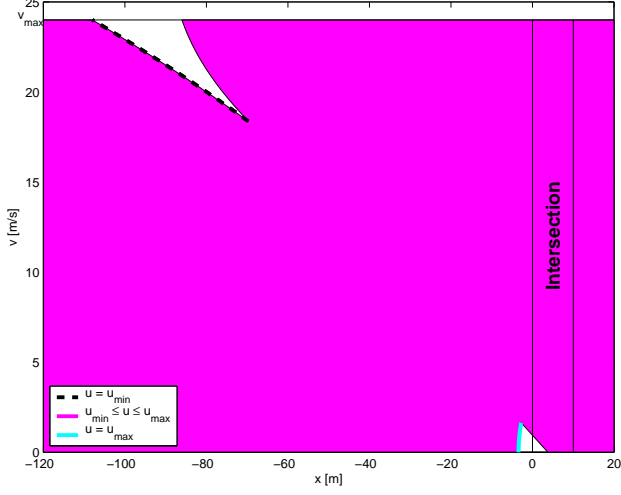


Figure 6: Regions of the state-space according to control action during the green interval:  $u = u_{\min}$ ,  $u \in [u_{\min}, u_{\max}]$ ,  $u = u_{\max}$ . Following these control restrictions guarantees that the car will never be caught in a red light violation.

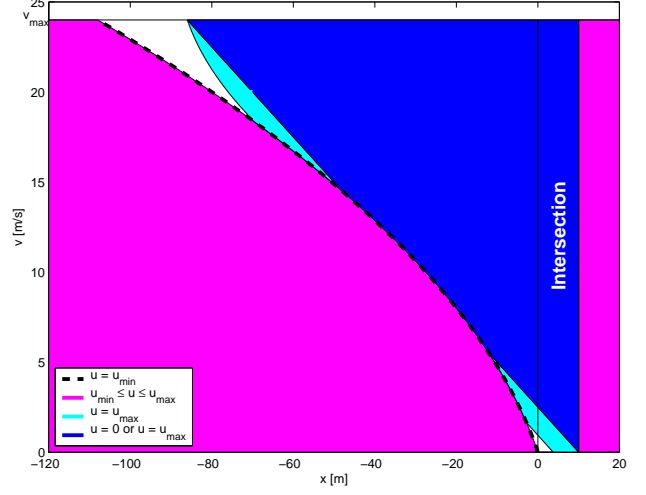


Figure 7: Regions of the state-space according to control action during the yellow interval:  $u = u_{\min}$ ,  $u \in [u_{\min}, u_{\max}]$ ,  $u = u_{\max}$ ,  $u = 0$  or  $u = u_{\max}$ .

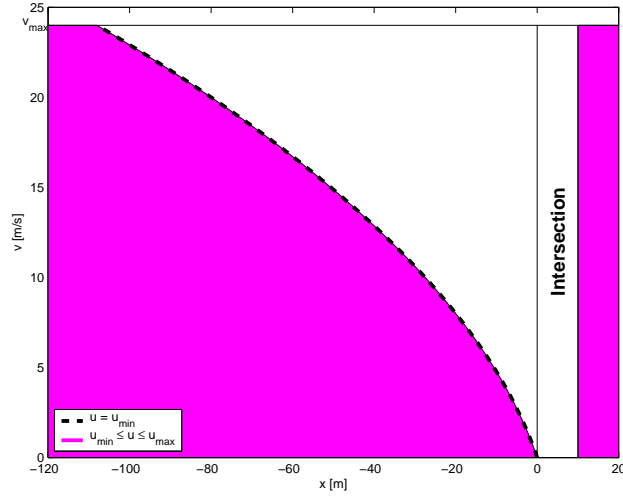


Figure 8: Regions of the state-space according to control action during the red interval:  $u = u_{\min}$ ,  $u \in [u_{\min}, u_{\max}]$ .

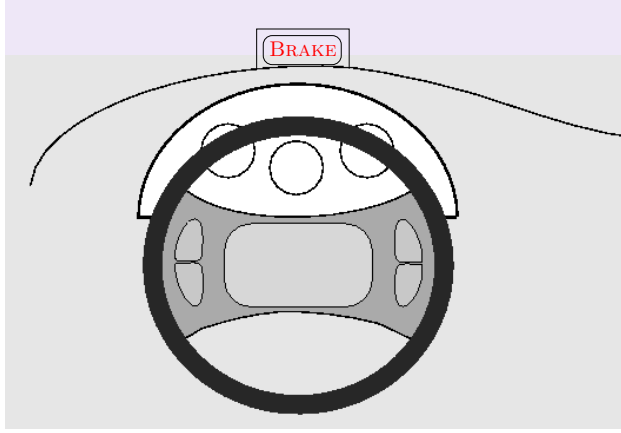


Figure 9: Add-on dashboard device.

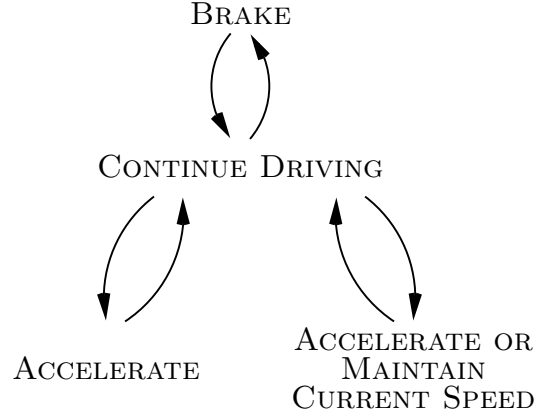


Figure 10: Interface indications for the dashboard device, as determined from Figures 6, 7, 8, and Table 1. The indication “UNSAFE” is not shown since unsafe operation will always be prevented if the user follows the indications displayed on the interface.

to drive through the intersection.

The methodology presented here guarantees that the interface, generated from analysis of the underlying continuous system, is correct. In general, the correctness of this interface is paramount: if the interface provides incomplete or incorrect information, the driver may not be able to avoid a red light violation. An incomplete or incorrect interface can appear nondeterministic to the user. Such interfaces are extremely problematic and dangerous: they confuse the user by sometimes providing correct information and sometimes providing wrong information, or by simply not providing enough information. As a result, the user is both misguided and confused [25].

In the yellow interval problem, we have generated a control law and determined the information content of an interface which guarantees the safety of the physical system. Thus, this kind of analysis tool, described here for the yellow interval example, can be used in the process of designing new interfaces. In the next section, we formulate a methodology to analyze human-automation systems which have both discrete and continuous dynamics.

### 3 Methodology

In this report, we propose a methodology to analyze a hybrid human-automation system based on well-developed tools in the realm of *hybrid system reachability analysis and controller synthesis* [18, 19, 17]. Given a hybrid model of the human-automation system (which includes how the user interacts with the system), we find a discrete representation of the hybrid model based on the hybrid system reachability result. Using this discrete representation, we can then design a correct interface for the hybrid system. In cases in which a proposed interface already exists, an existing and well-developed interface verification technique [11, 25] can be used to verify that the interface accurately represents the underlying system.

The hybrid human-automation system represents the underlying physical system as well as the mode-logic through which the user interacts with the physical system. While in general, all possible behaviors of the actual system can be described in the hybrid system model, often only a subset of these possibilities are included: this subset represents a *procedural* model. In many safety-critical systems, users (such as aircraft pilots or nuclear power technicians) perform tasks in highly structured ways. Here, we take advantage of this structure by modeling user-interaction with respect to a particular procedure.

We compute a discrete abstraction of the hybrid human-automation system using hybrid system reachability analysis. The methodology has three steps: first, a separation into smaller subsystems; second, a hybrid reachability analysis and controller synthesis; and third, a discrete abstraction based on the result of this analysis. We demonstrate this method with the yellow interval problem. We shall begin by formulating the problem mathematically.

#### 3.1 Mathematical Formulation of the Problem

We will consider two kinds of mathematical models in this problem: a hybrid model which incorporates both continuous and discrete state dynamics, and a discrete state model.

The hybrid model encapsulates the desired task or behavior that the user and the system should accomplish. We represent the system mathematically by the symbol  $H_{\text{procedure}}$ . The hybrid system is defined by the tuple  $H_{\text{procedure}} = (Q_{\text{procedure}}, \mathcal{X}_{\text{procedure}}, f_{\text{procedure}}, \Sigma_{\text{procedure}}, \mathcal{U}_{\text{procedure}}, R_{\text{procedure}})$ .  $Q_{\text{procedure}}$  represents the set of discrete states, or modes, in  $H_{\text{procedure}}$ ; with  $n$  modes, the cardinality of  $Q_{\text{procedure}}$  is  $|Q_{\text{procedure}}| = n$ . The function  $f_{\text{procedure}}$  is the set of continuous dynamics, which we will denote by the functions  $f_i$ ,  $\mathcal{U}_{\text{procedure}}$  is the set of all continuous control input vectors  $u_i$ , where  $i \in \{1, \dots, n\}$ . The hybrid model may be explained as follows: each mode  $q_i \in Q_{\text{procedure}}$  has dynamics  $\dot{x} = f_i(x, u_i)$ , in which the continuous state is  $x \in \mathcal{X}_{\text{procedure}}$  and the continuous

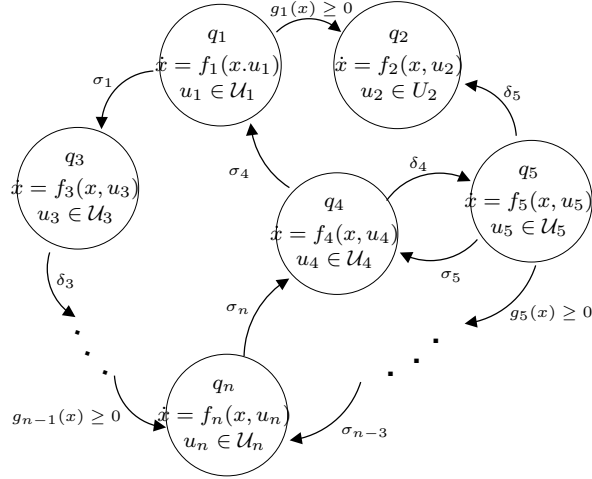


Figure 11: Example of a hybrid model. The naming convention for this report is that *user-controlled*, *disturbance*, and *automatic* switches are represented by  $\sigma_i$ ,  $\delta_i$ , and  $g_i(x) \geq 0$  (the state-based conditions under which the automatic transition occurs), respectively.

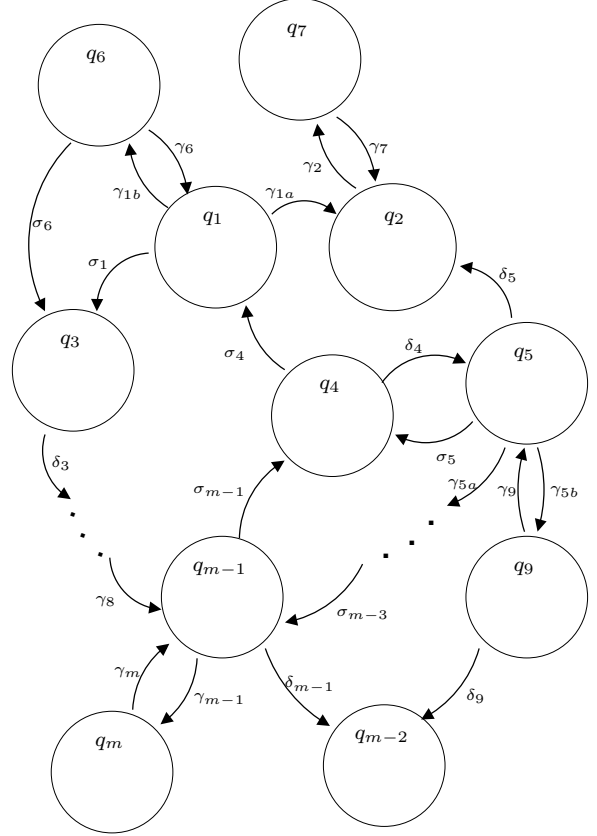


Figure 12: Example of a discrete model. The naming convention for this report is that *controlled* switches are represented by  $\sigma_i$ , and *uncontrolled* (disturbance and automatic) switches are represented by  $\delta_i$  and  $\gamma_i$ , respectively.

input  $u_i \in \mathcal{U}_i$  represents different control inputs in each mode. We assume that the continuous dynamics are fully observable (we can measure all elements of the state vector  $x$ ). The set of events which can enable or force the hybrid system to transition between modes is indicated by  $\Sigma_{\text{procedure}}$ . The transition function  $R_{\text{procedure}}$  dictates how the system switches according to discrete events  $\Sigma_{\text{procedure}}$ : for two modes  $q_a, q_b \in Q_{\text{procedure}}$ , a transition from  $q_a$  to  $q_b$  is possible when event  $\sigma \in \Sigma_{\text{procedure}}$  occurs if  $q_b \in R_{\text{procedure}}(q_a, \sigma)$ . The events in  $\Sigma_{\text{procedure}}$  fall into one of three categories: they can be *user-controlled*, *disturbance* (not controlled by the user or the automated system), or they can be *automatic* (determined by conditions on  $x$ ). In addition to the six elements accounted for in the tuple  $H_{\text{procedure}}$ , the set of hybrid states which the system can begin in (the initial set) is represented by  $(Q_{\text{procedure}}^0, \mathcal{X}_0)$ . An example of a hybrid system is shown in Figure 11.

The discrete state model is represented as:  $G_{\text{interface}} = (Q_{\text{interface}}, \Sigma_{\text{interface}}, R_{\text{interface}})$ , with modes  $Q_{\text{interface}}$ , events  $\Sigma_{\text{interface}}$ , and transition function  $R_{\text{interface}}$ . The set of initial modes is

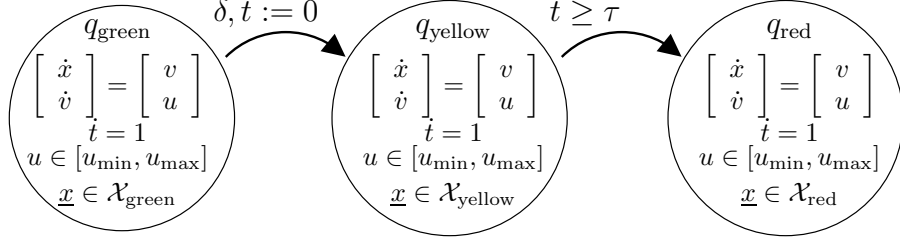


Figure 13: Hybrid model  $H_{\text{procedure}}$  for a car traveling through an intersection, where the input  $u$  is the car's acceleration, and  $\underline{x} := [x, v]$ .

$Q_{\text{interface}}^0$ . Events in  $\Sigma_{\text{interface}}$  fall into one of two categories: they are either *controlled* (initiated by the user) or *uncontrolled* (automatic or initiated by a disturbance). An example of a discrete system is shown in Figure 12 – we will use this discrete model to represent the interface of the system. Throughout this report, discrete events are interchangeably referred to as events, transitions, or switches.

In the yellow interval example, we model the car's dynamics and allowable ranges of operation during each color of the light. The hybrid model has modes  $Q_{\text{procedure}} = \{q_{\text{green}}, q_{\text{yellow}}, q_{\text{red}}\}$  and events  $\Sigma_{\text{procedure}} = \{\delta, g_1(\underline{x}, t)\}$ , where  $\delta$  represents a disturbance event (the light turning yellow), and  $g_1(\underline{x}, t) = t - \tau$  represents a switching surface which corresponds to the duration of the yellow interval. The transition function  $R_{\text{procedure}}$  determines the relationship between the modes and events through  $R_{\text{procedure}}(q_{\text{green}}, \delta) = q_{\text{yellow}}$ , and  $R_{\text{procedure}}(q_{\text{yellow}}, g_1(\underline{x}, t) \geq 0) = q_{\text{red}}$ . The dynamics in each mode are shown in the hybrid automaton of Figure 13. The state  $\underline{x} = [x, v]$  is bounded by  $\mathcal{X}_{\text{green}} = \mathbb{R} \times (0, v_{\text{max}}]$ ,  $\mathcal{X}_{\text{yellow}} = \{(\mathbb{R} \setminus 0) \times (0, v_{\text{max}}]\} \cup \{0 \times [0, v_{\text{max}}]\}$ , and  $\mathcal{X}_{\text{red}} = \{\mathbb{R}^- \times (0, v_{\text{max}}]\} \cup \{0 \times (0, v_{\text{max}}]\} \cup \{[L, \infty) \times (0, v_{\text{max}}]\}$ , where  $\mathbb{R}$  is the set of real numbers. We are interested in positive speeds at or below the speed limit only, except right at the intersection ( $x = 0$ ) during the yellow and red lights. Time is indicated through a timer function  $\dot{t} = 1$  in each mode, with  $t \in \mathbb{R}$ . The input  $u$  is bounded by the interval  $\mathcal{U} = [u_{\text{min}}, u_{\text{max}}]$  in each mode, which accounts for maximum deceleration  $u_{\text{min}}$  and maximum acceleration  $u_{\text{max}}$ . The initial mode is  $Q_{\text{procedure}}^0 = \{q_{\text{green}}\}$ , since we assume the driver is traveling along the expressway at a constant velocity when the yellow light appears.

In the following sections, we show how to form a discrete abstraction of the hybrid procedural model using a modified reachability analysis.

### 3.2 Step 1: Separation into Hybrid Subsystems

The first step in forming a discrete abstraction is to separate the hybrid procedural model,  $H_{\text{procedure}}$ , across user-controlled switches. Thus, within each subsystem, all of the transitions are uncontrolled

(disturbance and automatic) transitions. This will prepare us for the next step, a hybrid reachability analysis and controller synthesis to be performed on each hybrid subsystem.

Every hybrid subsystem has an initial mode defined either by a mode in  $Q_{\text{procedure}}^0$  or by the mode that a user-controlled switch leads to. We say a set of modes is *automatic-reachable* if the modes are reachable from a given initial mode with automatic or disturbance transitions only. The rest of the hybrid subsystem is determined by augmenting the subsystem containing this initial mode with all modes automatic-reachable from the initial mode. The user-controlled transitions are not included in the subsystems, since the user has the final authority over the switch, and therefore guarantees of safety lie with the user. (Implications of this will be made clear in Section 3.4.)

For the yellow interval problem, the hybrid mode has no user-controlled switches. More general (and often more complicated) hybrid models will separate into hybrid subsystems with more than one mode, and which might even overlap. To clarify, imagine a generic hybrid model with  $n$  user-controlled switches, and  $m = |Q_{\text{procedure}}^0|$  initial modes, where  $|\cdot|$  indicates cardinality of a given set. This hybrid procedural model could have at most  $(m + n)$  hybrid subsystems, since one hybrid subsystem can result from each user-controlled switch, and one hybrid subsystem can result from each initial mode.

For example, the hybrid system in Figure 14 has  $n_0 = 1$  initial modes ( $q_1$ ) and  $n = 5$  user-controlled switches ( $\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5$ ). It should have  $n + n_0 = 6$  or fewer hybrid subsystems. We separate this model into five hybrid subsystems:  $H_A$  consists of the modes automatic-reachable from initial mode  $q_1$ ,  $H_B$  consists of modes automatic-reachable from the user-controlled switch  $\sigma_5$  into  $q_{13}$ ,  $H_C$  consists of modes automatic-reachable from switches  $\sigma_1$  into  $q_5$ ,  $H_D$  consists of modes automatic-reachable from  $q_8$ , and  $H_E$  consists of modes automatic-reachable from  $q_{11}$ . This model contains five (but not six) hybrid subsystems since the user-controlled switches  $\sigma_2$  and  $\sigma_3$  both lead to the same mode,  $q_8$ .

### 3.3 Step 2: Hybrid Reachability Analysis and Controller Synthesis

Hybrid reachability analysis provides us with a mathematical guarantee of *safety* to within the limits of the hybrid model. We can define safety mathematically as the system's ability to remain within an allowable region of the hybrid state-space. Physical constraints on the system can be incorporated in this manner: for example, if the car on the expressway must obey the speed limit, the car's velocity must not exceed 24 m/s ( $v \in (0, 24]$ ). Since physical systems are also subject to input constraints (due to actuator saturation, for example), we have bounded control authority to keep the system within its allowable region of operation. In the yellow light dilemma, the driver's

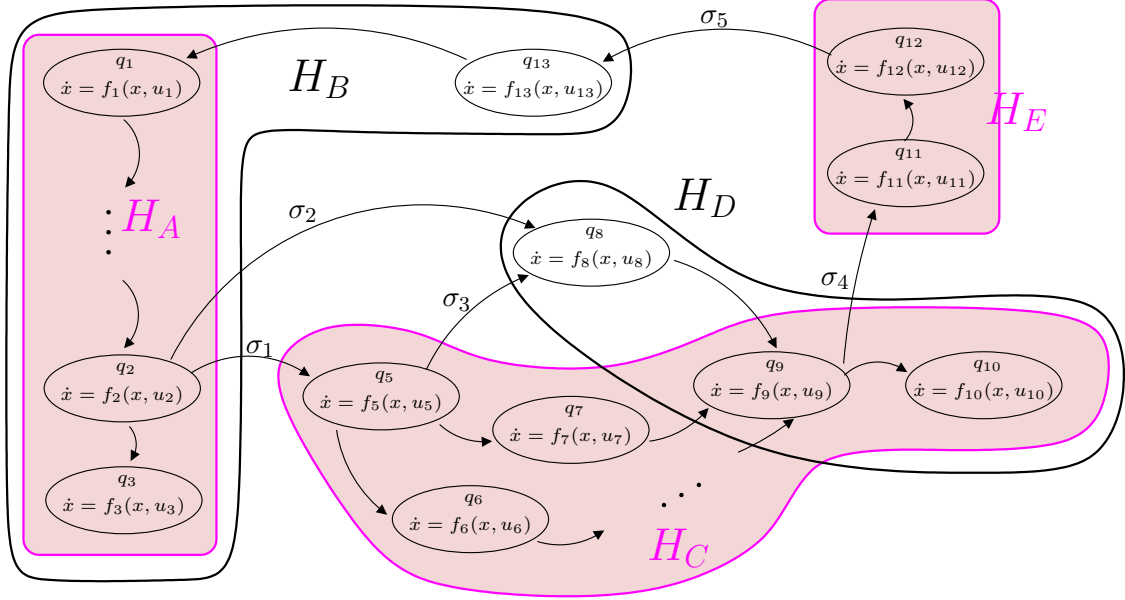


Figure 14: Hybrid procedural automaton separated into hybrid subsystems  $H_A, H_B, H_C, H_D, H_E$ . User-controlled switches are indicated by  $\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5$ . (All other switches are uncontrolled.) The initial mode is  $Q_{\text{procedure}}^0 = q_1$ . Note that the hybrid subsystems overlap. Some of the hybrid subsystems are shaded for clarity.

car has a maximum deceleration which represents the capabilities of the physical system: the car's brakes simply cannot provide more braking force than  $u_{\min}$ .

In order to remain within the *allowable region* of operation, which we denote  $\mathcal{W}_0$ , we are often restricted to remain within an even smaller region: a subset of the allowable region of operation. (See Figure 15.) This is because, for some states in  $\mathcal{W}_0$ , there may be no control input guaranteed to keep trajectories of the system inside  $\mathcal{W}_0$ . The hybrid reachability analysis provides us with the largest subset within  $\mathcal{W}_0$  in which we can guarantee the system will always remain – this is the *maximal controlled invariant set*  $\mathcal{W} \subseteq \mathcal{W}_0$ . This is the *safe region* – the region for which the system always has an input which will keep the system within the safe area. System trajectories can only exit  $\mathcal{W}$  along its boundary, thus, inside  $\mathcal{W}$  any control  $u \in \mathcal{U}$  can be used, but along the boundary of  $\mathcal{W}$ , a set of controllers  $u = u^*(x)$  which forces the system to remain within  $\mathcal{W}$  must be used. If any other control law is used along the boundary, the system will leave the safe region  $\mathcal{W}$ , and eventually will leave the allowable region of operation  $\mathcal{W}_0$ . If the system begins within  $\mathcal{W}$ , it can always remain within  $\mathcal{W}$ , and therefore within  $\mathcal{W}_0$ . (If the system began in  $\mathcal{W}_0 \setminus \mathcal{W}$ , we would not be able to guarantee that it will always remain in  $\mathcal{W}_0$ .)

How do we formulate  $\mathcal{W}_0$  and use it to obtain the safe region  $\mathcal{W}$ , as well as the control law to enforce safety,  $u^*(x)$ ? We begin with the formulation of a *cost function*. Constraints on the system's continuous states and inputs are represented through the use of a cost function  $J(x, t)$  which we

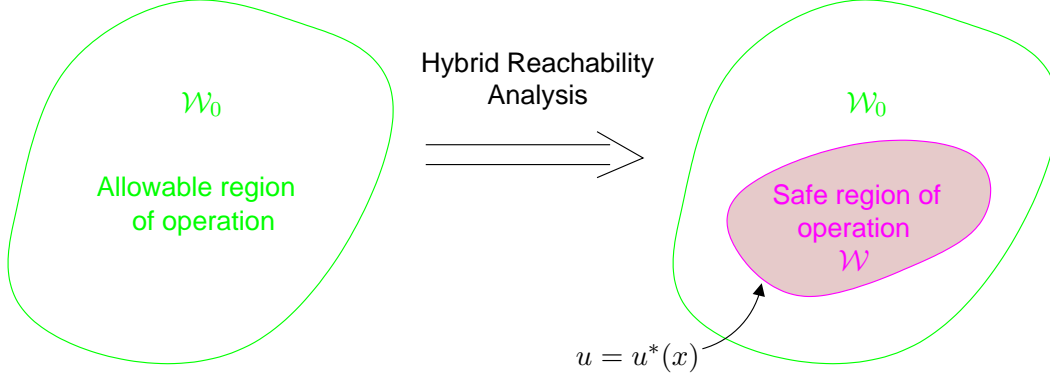


Figure 15: Hybrid reachability analysis and controller synthesis determines the 1) safe region of operation  $\mathcal{W} \subseteq \mathcal{W}_0$ , and 2) the set of control laws  $u = u^*(x)$  which must be applied on the boundary of  $\mathcal{W}$  in order for the state of the system to remain within  $\mathcal{W}$ , and therefore within the allowable region  $\mathcal{W}_0$ .

will evolve backwards in time ( $t \leq 0$ ). The value of this cost function initially is  $J(x, 0) = J_0(x)$ , where  $J_0(x)$  indicates the boundary of the allowable region of operation  $\mathcal{W}_0$ . The values of  $x$  for which  $J_0(x) = 0$  constitute the boundary of the allowable region; the sign of  $J_0(x)$  for other values of  $x$  indicates if the system is inside or outside of the allowable region. Using standard notation to indicate the interior  $(\mathcal{W}_0)^\circ$ , boundary  $\partial\mathcal{W}_0$ , and exterior (complement)  $(\mathcal{W}_0)^c$  of the set  $\mathcal{W}_0$ ,

$$J_0(x) > 0 \quad \text{if } x \in (\mathcal{W}_0)^\circ \quad (1)$$

$$J_0(x) = 0 \quad \text{if } x \in \partial\mathcal{W}_0 \quad (2)$$

$$J_0(x) < 0 \quad \text{if } x \in (\mathcal{W}_0)^c. \quad (3)$$

We can now indicate the allowable region of operation as  $\mathcal{W}_0 = \{x \in \mathcal{X} \mid J_0(x) \geq 0\}$ .

To see how to apply this calculation, we examine the yellow interval dilemma. To avoid being *unsafe*, or in the intersection during a red light, the initial cost function represents the car coasting safely past the intersection at the start of the red light:  $J_0(x) = x - L$ . Positive values of  $J_0$  occur when the car's position  $x$  is past the far side of the intersection. During a red light, if the car is coasting, the allowable state-space region is  $x > L$ , which is where  $J_0(x) > 0$ . Negative values of  $J_0$  occur when the car has yet to completely cross the intersection, and  $J_0 = 0$  when the car is just at the boundary of the intersection, at  $x = L$ . However, we know from Section 2 that this boundary is dependent on time – on the duration of the yellow interval.

The cost function  $J$  varies over time and over values of  $x$ : its boundary  $J(x, t) = 0$  evolves as  $x$  evolves according to the equations of motion,  $\dot{x} = f(x, u)$ . We know that along the boundary the control input  $u$  must have certain properties: to enforce safety the set of control laws  $u^*(x)$  is synthesized as  $\mathcal{W}$  is determined. This control law must be enforced when the state of the system

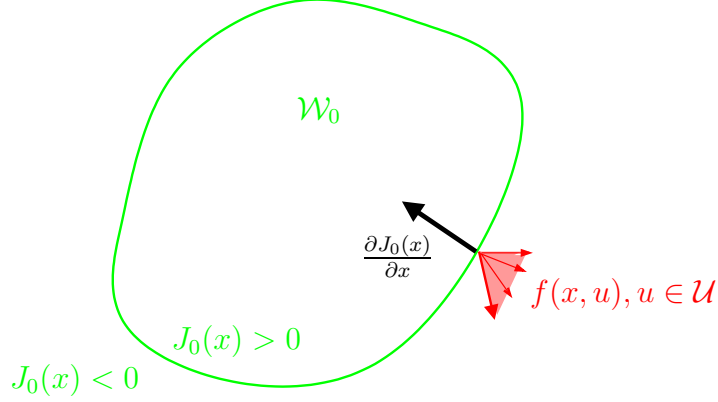


Figure 16: Initial cost function  $J_0(x)$  and initial Hamiltonian  $H(x, \frac{\partial J_0(x)}{\partial x})$ . The particular control  $u \in \mathcal{U}$  chosen for the Hamiltonian (see definition in text) is bolded.

reaches the boundary of  $\mathcal{W}$ , in order to be able to guarantee that the state of the system will always remain within  $\mathcal{W}$ .

To find  $\mathcal{W}$  within a given discrete mode, we first formulate the Hamiltonian  $H(x, \frac{\partial J(x,t)}{\partial x}) = \max_{u \in \mathcal{U}} \frac{\partial J(x,t)}{\partial x}^T f(x, u)$  for a nonlinear system with dynamics  $\dot{x} = f(x, u)$ ,  $x \in \mathbb{R}^n$ ,  $u \in \mathcal{U} \subseteq \mathbb{R}^m$ . The control chosen through this maximization effectively tries to make  $\mathcal{W}$  as large as possible. Geometrically, for the initial cost function  $J(x, 0) = J_0(x)$ , the initial Hamiltonian is the inner product of the inward-pointing normal of  $\mathcal{W}_0$  (that is,  $\frac{\partial J_0(x)}{\partial x}$ ), and the function in the direction closest to this normal (see Figure 16). The cost function  $J(x, t)$  evolves backwards in time according to the partial differential equation (PDE)

$$\frac{\partial J^*(x, t)}{\partial t} = - \min \left\{ H \left( x, u^*, \frac{\partial J^*(x, t)}{\partial x} \right), 0 \right\} \quad (4)$$

The minimization between  $H(x, \frac{\partial J(x,t)}{\partial x})$  and 0 ensures that once a trajectory passes through an unsafe region, it cannot later become safe. If the cost function  $J(x, t)$  converges to  $J^*(x)$  as  $t \rightarrow -\infty$ , the safe region of operation is defined as

$$\mathcal{W} = \{x \in \mathcal{W}_0 \mid J^*(x) \geq 0\} \quad (5)$$

The control law synthesized from this calculation

$$u^*(x) = \{u \in \mathcal{U} \mid \frac{\partial J^*(x)}{\partial x}^T f(x, u) \geq 0\} \quad (6)$$

must be applied along the boundary of the safe region, for  $x \in \partial\mathcal{W}$ , in order guarantee that the region is invariant.

In the yellow interval problem, we complete one hybrid reachability analysis. This results in safe regions for each phase of the light:  $\mathcal{W}_{\text{green}}$ ,  $\mathcal{W}_{\text{yellow}}$ , and  $\mathcal{W}_{\text{red}}$ . During the yellow light, the user's options (stopping before the intersection, coasting through it, or accelerating through it) result in regions  $\mathcal{W}_{\text{brake}}$ ,  $\mathcal{W}_{\text{coast}}$ ,  $\mathcal{W}_{\text{accel}}$ :  $\mathcal{W}_{\text{yellow}} = \mathcal{W}_{\text{brake}} \cup \mathcal{W}_{\text{coast}} \cup \mathcal{W}_{\text{accel}}$ . Figures 1 and 2 show the results of the hybrid reachability analysis: the shaded region of Figure 1 is  $\mathcal{W}_{\text{brake}}$ , the region in which safe braking is guaranteed. The darker shaded region of Figure 2 is  $\mathcal{W}_{\text{coast}}$ , the region in which safe coasting is guaranteed, and the lighter shaded region is  $\mathcal{W}_{\text{accel}}$ , the region from which the car can safely cross the intersection while accelerating at the maximum acceleration. In Figure 2, the dark shaded region is the set of values of  $\underline{x} = [x, v]$  for which  $J_{\text{coast}}^*(\underline{x}) \geq 0$ , where  $J_{\text{coast}}^*(\underline{x}) = x - L + v\tau$ . (Recall the duration of the yellow interval is  $\tau$ .) On the boundary of  $\mathcal{W}_{\text{coast}}$ ,  $J_{\text{coast}}^*(\underline{x}) = 0$ , and outside of  $\mathcal{W}_{\text{coast}}$ ,  $J_{\text{coast}}^*(\underline{x}) < 0$ . In  $\mathcal{W}_{\text{coast}}$ , any control which satisfies  $u \geq 0, \dot{u} \geq 0$  can be used. The light shaded region  $\mathcal{W}_{\text{accel}}$  in addition to the states included in  $\mathcal{W}_{\text{coast}}$  are those combinations of  $x$  and  $v$  for which acceleration at  $u = u_{\text{max}}$  will allow the car to drive through the intersection before the red light appears. In this case,

$$J_{\text{accel}}^*(\underline{x}) = \begin{cases} x - L + v\tau + u_{\text{max}}(\tau - \Delta)^2/2 & \text{for } v \in (0, \bar{v}] \\ x - L + v_{\text{max}}\tau + \Delta(v - v_{\text{max}}) - \frac{(v - v_{\text{max}})^2}{2u_{\text{max}}} & \text{for } v \in (\bar{v}, v_{\text{max}}] \end{cases} \quad (7)$$

with  $\bar{v} = v_{\text{max}} - u_{\text{max}}(\tau - \Delta)$ . Inside  $\mathcal{W}_{\text{brake}}$ , any control in the allowable range  $\mathcal{U}$  may be used, and on the boundary of  $\mathcal{W}_{\text{brake}}$ ,  $u^* = u_{\text{min}}$  must be enforced to guarantee that the system will not leave the safe region  $\mathcal{W}_{\text{brake}}$ . We can find a closed-form solution  $J_{\text{brake}}^*(\underline{x}) = -x - v\Delta + \frac{v^2}{2u_{\text{min}}}$  for  $x \in \mathcal{X}_{\text{yellow}}$ ; notice that this parabola is truncated at  $v_{\text{max}}$  for  $x \leq -v_{\text{max}}\Delta + \frac{v_{\text{max}}^2}{2u_{\text{min}}}$ .

For the red light,  $\mathcal{W}_{\text{red}} = \mathcal{W}_{\text{brake}} \cup \{[L, \infty) \times (0, v_{\text{max}}]\}$ . During the green light, propagating the boundaries of  $\mathcal{W}_{\text{yellow}}$  results in a slightly smaller set, as shown in Figure 4. Since the car is not allowed to come to a full stop ( $v = 0$ ) during the green light, we cannot allow the car to be “trapped” in the lower unshaded region. The optimal cost function for the changed boundary (along which  $u = u_{\text{max}}$  must be enforced) is  $J_{\text{green}}(\underline{x}) = -x + \tilde{x} + \frac{v^2 - \tilde{v}^2}{2u_{\text{max}}}$ , with  $\tilde{v}$  the smaller solution to  $0 = \frac{1}{2u_{\text{min}}}\tilde{v}^2 + (\tau - \Delta)\tilde{v} + (\tau - \Delta)^2u_{\text{max}}/2 - L$ , and  $\tilde{x} = \frac{\tilde{v}^2}{2u_{\text{min}}} - \Delta\tilde{v}$ .

For more complicated hybrid systems, with more than one mode, this analysis is extended to include the option of switching to other modes. The hybrid reachability algorithm, introduced in [17], accounts for the effect of mode changes on the continuous evolution of  $J(x, t)$ . Figure 17 shows the algorithm graphically for two modes. Evolution in one mode,  $q_1$  does not evolve independently of the dynamics of  $q_2$  if there is a switch which can occur from  $q_1 \rightarrow q_2$ . For example, if there is an allowable region  $(\mathcal{W}_1)_0$  in mode  $q_1$  and another allowable region  $(\mathcal{W}_2)_0$  in mode  $q_2$ , and  $q_1 \rightarrow q_2$  through an automatic transition across a switching boundary  $g(x) = 0$ , the safe regions  $\mathcal{W}_1$  and  $\mathcal{W}_2$  cannot be computed independently. For the system shown in Figure 17, if the switch is ignored, the safe regions in  $q_1$  and  $q_2$  are  $\mathcal{A}$  and  $\mathcal{D}$ , respectively. However, the switch effectively increases

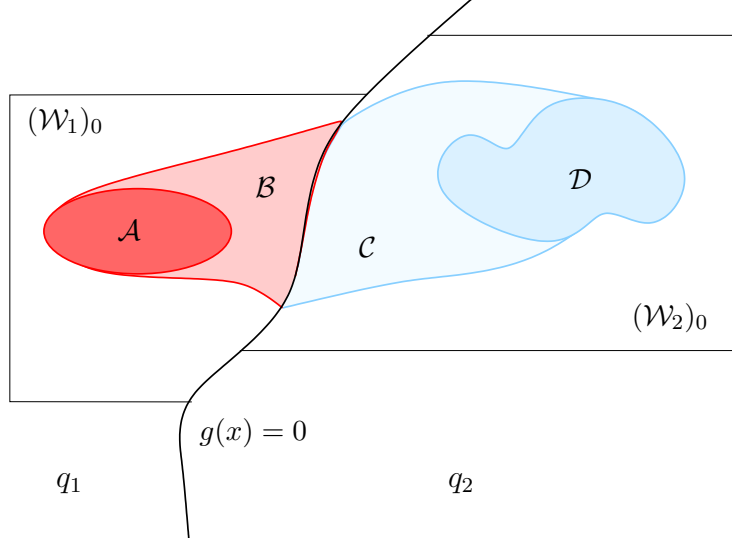


Figure 17: The hybrid reachability algorithm accounts for switches to other modes. Note when the switch across  $g(x) = 0$  is accounted for, safe regions  $\mathcal{W}_1 = \mathcal{A} \cup \mathcal{B}$  and  $\mathcal{W}_2 = \mathcal{C} \cup \mathcal{D}$  are larger than when determined in isolation in modes  $q_1$  and  $q_2$ , respectively. ( $\mathcal{A}$  is the safe region in  $q_1$ , and  $\mathcal{D}$  is the safe region in  $q_2$  if the switch from  $q_1$  to  $q_2$  is not allowed.)

the safe region by allowing the system to switch to safety in the next mode. In this case, the safe region is  $\mathcal{W}_1 = \mathcal{A} \cup \mathcal{B}$  in mode  $q_1$  and  $\mathcal{W}_2 = \mathcal{C} \cup \mathcal{D}$  in mode  $q_2$ .

In general, closed-form solutions (such as the ones in the yellow interval dilemma) are not possible for  $\mathcal{W}$ . Additionally, when the safe region is represented as the intersection of various cost functions, simply integrating backwards in the presence of continuous inputs and disturbances may not yield the correct solution, since intersections of the evolving boundaries may not be analytically determined. Accurate computational methods to determine  $\mathcal{W}$  are an important topic of research, and offer solutions when analytic solutions are not possible or tractable. Computational algorithms have been developed to find the boundary of  $\mathcal{W}$  using level-set methods [19]. Further details of this algorithm (for a full hybrid system) can be found in [17], and information regarding computational implementation can be found in [33, 20, 26].

### 3.4 Step 3: Abstraction: Hybrid $\rightarrow$ Discrete

We create a discrete abstraction based on the results of this analysis. In many complex human-automation systems, the user has control over discrete switches, but the continuous control to maintain safety within each continuous mode is automatically enforced. Within each hybrid subsystem, the controller for safety  $u^*(x)$  is enforced. (This is not the case for the car example, in which the user must enact various simple continuous control laws in order to remain safe, but will be demonstrated in Section 4.)

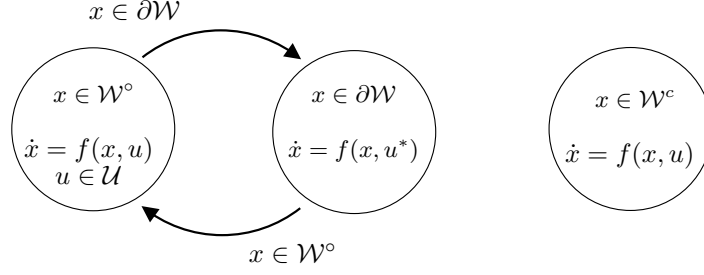


Figure 18: Hybrid automaton  $H^*$ .

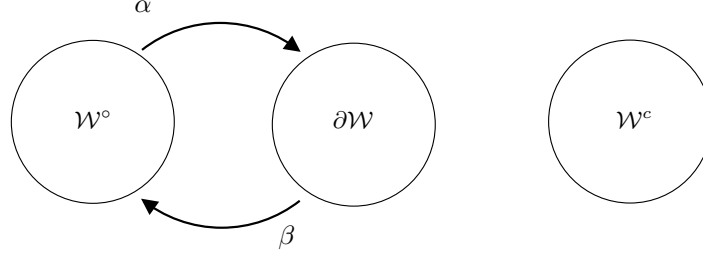


Figure 19: Abstracted discrete automaton  $G^*$ .

We assume that if the system begins within a safe region of operation, the only way the system can become unsafe is through user-controlled discrete switches. Therefore, if the user enters a hybrid subsystem safely, the system will remain safe until the next user-controlled discrete switch. The system is also unsafe if it begins in an unsafe region of operation. (This is the case in the yellow interval problem, when the light turns yellow and the car is in region (7) of Figure 3.)

The goal of this section is to abstract from the hybrid model  $H$  a discrete system  $G^*$ , which can be used in interface analysis, verification, and design methods. We begin with the simplest hybrid system: one with only one mode. For a generic, one-mode hybrid system  $H$ , implementing the control law  $u^*(x)$  results in a three-mode hybrid system  $H^*$  as shown in Figure 18. Switching between modes is dictated by evolution of the continuous state  $x$ . No switches exist from  $x \in \partial\mathcal{W}$  to  $x \in \mathcal{W}^c$  since, as a result of the hybrid reachability analysis and controller synthesis, the control law  $u = u^*(x)$  enforces safety when  $x \in \partial\mathcal{W}$ . We form the discrete system  $G^*$  from  $H^*$  by partitioning the state space into the interior, boundary, and complement of  $\mathcal{W}$ :  $x \in \{\mathcal{W}^\circ, \partial\mathcal{W}, \mathcal{W}^c\}$ . Since  $\mathcal{W}$  consists of its interior and its boundary, note that  $\mathcal{W} = \mathcal{W}^\circ \cup \partial\mathcal{W}$ , and  $\mathcal{W} \cap \mathcal{W}^c = \{\emptyset\}$ . The discrete system  $G^* = (Q^*, \Sigma^*, \delta^*)$  has modes  $Q^* = \{\mathcal{W}^\circ, \partial\mathcal{W}, \mathcal{W}^c\}$ , which correspond to the continuous regions of operation in  $H^*$ . The continuous state-based transitions of  $H^*$  become the events  $\Sigma^* = \{\alpha, \beta\}$  in  $G^*$ . The transition function  $R^*$  is depicted graphically in Figure 19.

We now consider a hybrid system  $H$  with two modes  $Q = \{q_i, q_j\}$  and an automatic transition  $\Sigma = \{\gamma\}$  which takes the system from  $q_i$  to  $q_j$ . The hybrid reachability analysis results in a hybrid system  $H^*$ , which we abstract to  $G^*$  in Figure 20.  $G^*$  has events  $\Sigma^* = \{\alpha_i, \beta_i, \alpha_j, \beta_j, \gamma\}$  and modes

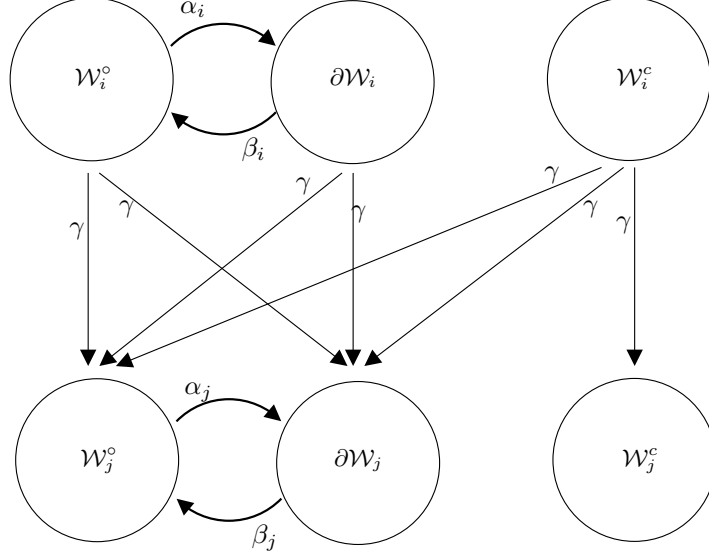


Figure 20: Nondeterministic abstraction of generic two-mode hybrid automaton  $q_i \xrightarrow{\gamma} q_j$ .

$Q^* = \{W_i^o, \partial W_i, W_i^c, W_j^o, \partial W_j, W_j^c\}$ . Because of the hybrid reachability analysis, if the system begins in a safe region ( $W_i^o$  or  $\partial W_i$ ) then  $\gamma$  will keep the system in a safe region ( $W_j^o$  or  $\partial W_j$ ). However, there is no guarantee of safety if the system begins in the unsafe region  $W_i^c$ . Notice that  $G^*$  is nondeterministic:  $W_i^o \xrightarrow{\gamma} \{W_j^o, \partial W_j\}$ ,  $\partial W_i \xrightarrow{\gamma} \{W_j^o, \partial W_j\}$ , and  $W_i^c \xrightarrow{\gamma} \{W_j^o, \partial W_j, W_j^c\}$ , but that the nondeterminism does not create ambiguity with respect to safety: states which are initially safe will always remain safe.

Consider the same hybrid system, but with one user-controlled transition  $\Sigma = \{\sigma\}$  from  $q_i$  to  $q_j$ . Because  $\sigma$  is user-controlled, we can make no guarantees of safety in the new mode  $q_j$ : even if the user starts within a safe region ( $W_i^o$  or  $\partial W_i$ ), the user could switch the system into an unsafe state ( $W_j^c$ ). We partition the state space according to the intersection of  $W_i$  and  $W_j$ , as shown in Figure 21. This results in  $3^2 = 9$  regions  $x \in \{(W_i^o \cap W_j^o), (W_i^o \cap \partial W_j), (W_i^o \cap W_j^c), (\partial W_i \cap W_j^o), \dots, (W_i^c \cap W_j^c)\}$  for  $q_i$ . When the user applies  $\sigma$ , the system transitions deterministically from  $q_i$  into  $q_j$ . Figure 21 shows the deterministic discrete system  $G^*$ . Notice that this abstraction results in a discrete system with far more modes: it has  $3 + 3^2 = 12$  modes, while the nondeterministic system in Figure 20 has  $3 + 3 = 6$  modes.

When more than one user-controlled switch is possible from a given mode the state-space must be further partitioned. For example, examine  $q_2$  in Figure 14: the partitioning of the state-space in mode  $q_2$  must not only account for the state-space in mode  $q_5$ , but also the state-space of mode  $q_8$ . To account for all possible user-actions, we define for a mode  $q_i$  the set of *user-reachable* modes

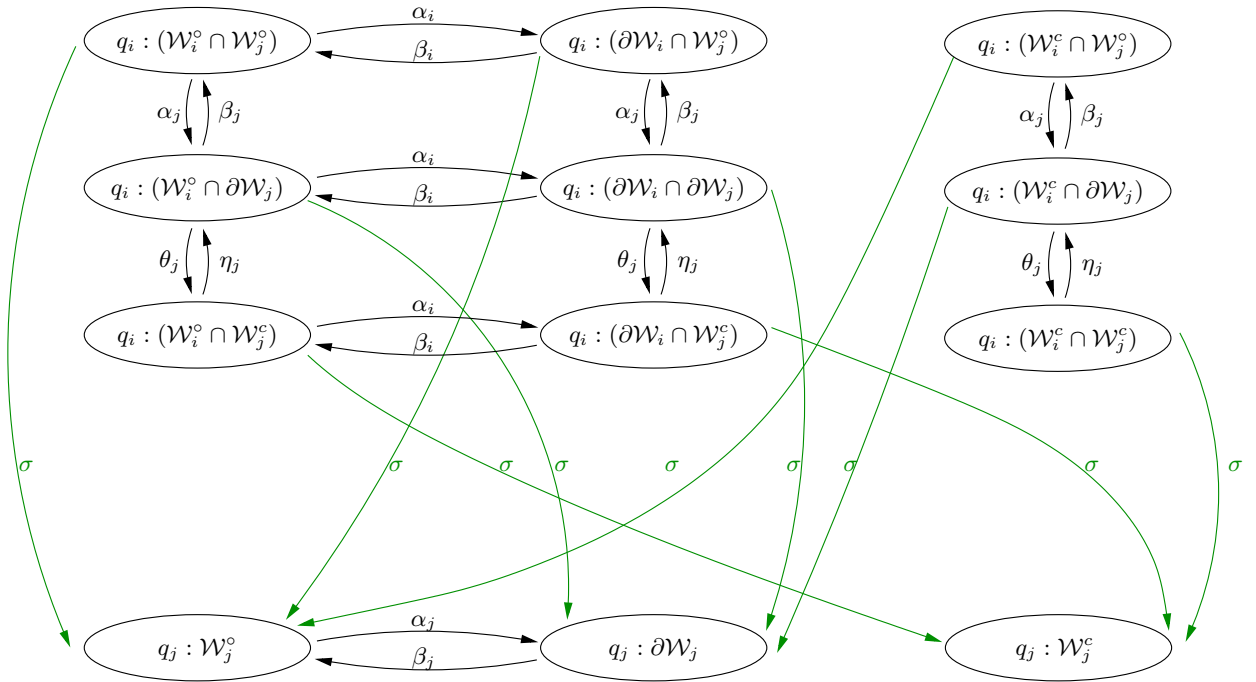


Figure 21: Deterministic abstraction  $G^*$  of generic two-mode hybrid automaton  $q_i \xrightarrow{\sigma} q_j$ . The notation used indicates the mode ( $q_i$  or  $q_j$ ), followed by the state-space partition the system is in. Note the transitions  $\theta_j, \eta_j$ , which account for continuous-state based transitions between the boundary and exterior of mode  $q_j$ , respectively, are hidden in the nondeterministic abstraction (Figure 20).

$P(q_i)$  as all distinct modes reachable through a user-controlled transition:

$$P(q_i) \triangleq \{q_k \mid R(q_i, \bar{\sigma}) = q_k\} \quad (8)$$

where  $\bar{\sigma}$  is the set of all user-controlled events possible from  $q_i$ . The cardinality  $n_i = |P(q_i)|$  determines the number of modes in  $G^*$  abstracted from one hybrid mode of  $H$ . The state-space in mode  $q_i$  must be partitioned into  $3^{n_i+1}$  regions, and the state-space in mode  $R(q_i, \sigma)$ ,  $\sigma \in \bar{\sigma}$ , must be partitioned into  $3^{(|P(R(q_i, \sigma))|+1)}$  regions. In Figure 14,  $P(q_2) = \{q_5, q_8\}$ , and since  $n_2 = 2$ , the state-space of  $q_2$  must be partitioned into  $3^{(2+1)} = 27$  discrete modes;  $P(q_5) = \{q_8\}$ , so the state-space of  $q_5$  will be partitioned into  $3^2$  regions; and  $P(q_8) = \{\emptyset\}$ , so the state-space of  $q_8$  will be partitioned into  $3^1$  regions.

We can now consider the more general case: a  $r$ -mode hybrid subsystem  $H = (Q, \mathcal{X}, f, \Sigma, u, R)$  with  $n_{\text{exit}}$  modes from which the user can manually exit the subsystem  $H$ . After applying the control law for safety  $u = u^*(x)$ , we obtain the safe region of operation  $\mathcal{W} \subseteq Q \times \mathcal{X}$ . The discrete system  $G^* = (Q^*, \Sigma^*, R^*)$  is formed by abstracting the hybrid subsystem, based on a state-space partition of  $\mathcal{W}$  in each mode  $q \in Q$ . For each mode  $q \in Q$ , we partition the state-space into  $P(q)$  regions. Those modes with no user-reachable modes (i.e. modes  $q$  for which  $P(q) = \{\emptyset\}$ ) will have three regions; the  $n_{\text{exit}}$  modes before user-controlled transitions will be partitioned according to their intersection with user-reachable modes outside of the subsystem  $H$ . Each region is relabeled as a discrete mode  $q^* \in Q^*$ . By construction,  $R^*$  accounts not only for the transitions in  $R$  (with labels appropriate for a discrete system), but also for transitions between the partitions which create  $Q^*$ . In general,  $R^*$  will be nondeterministic, but unambiguous with respect to transitions to unsafety.

We combine all of the abstracted hybrid subsystems into a single discrete system  $G_{\text{procedure}}^*$ , through the user-controlled switches. This abstraction makes safety unambiguous: it assumes that where possible, safety is guaranteed (across automatic and disturbance transitions); where safety cannot be guaranteed (across user-controlled transitions), the consequences of the user's actions with respect to system safety are deterministic. Across automatic transitions, less refinement in the state-space partition is necessary, since the system cannot become unsafe through an automatic transition. Across user-controlled transitions, more refinement is necessary in order to determine how the user, through the user's possible discrete actions, will affect system safety.

As an example, a portion of a hybrid system  $H_{\text{procedure}}$  is shown across two of its hybrid subsystems in Figure 22. Figure 23 shows the abstracted model  $G_{\text{procedure}}^*$  for this same portion. The mode before the switch is abstracted deterministically, but other modes within the hybrid subsystems are then abstracted nondeterministically.

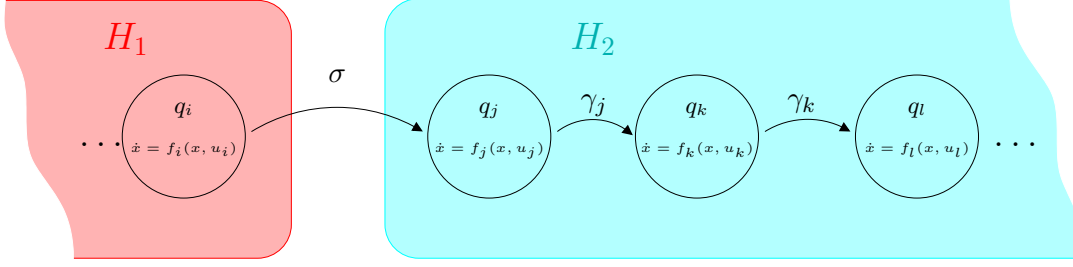


Figure 22: Portion of hybrid system  $H_{\text{procedure}}$  which displays two hybrid subsystems,  $H_1$  and  $H_2$ .

Notation conventions are established in Figures 22 and 23. Hybrid reachability analysis results in  $\mathcal{W}_i$  for each hybrid subsystem  $H_i$ , where  $\mathcal{W}_i \subseteq Q_i \times \mathcal{X}$ . For example,  $\mathcal{W}_2 \subseteq \{q_j, q_k, q_l, \dots\} \times \mathcal{X}$ . Since we need to examine modes within  $\mathcal{W}_i$  to properly create the discrete abstraction, we use  $\mathcal{W}_i^{q_j}$  to indicate the portion of  $\mathcal{W}_i$  in mode  $q_j \in Q_i$ . For example, to indicate the interior, boundary, and exterior of  $\mathcal{W}_2$  in mode  $q_k \in Q_2$ , we write  $(\mathcal{W}_2^{q_k})^\circ, \partial\mathcal{W}_2^{q_k}, (\mathcal{W}_2^{q_k})^c$ . The switches in  $G_{\text{procedure}}^*$  are labeled according the following convention:

- $\alpha_j$  occurs when  $x$  evolves from the *interior* of  $\mathcal{W}_i^{q_j}$  to its *boundary*  $((\mathcal{W}_i^{q_j})^\circ \text{ to } \partial\mathcal{W}_i^{q_j})$ .
- $\beta_j$  occurs when  $x$  evolves from the *boundary* of  $\mathcal{W}_i^{q_j}$  to its *interior*  $(\partial\mathcal{W}_i^{q_j} \text{ to } (\mathcal{W}_i^{q_j})^\circ)$ .
- $\theta_j$  occurs when  $x$  evolves from the *boundary* of  $\mathcal{W}_i^{q_j}$  to its *exterior*  $(\partial\mathcal{W}_i^{q_j} \text{ to } (\mathcal{W}_i^{q_j})^c)$ .
- $\eta_j$  occurs when  $x$  evolves from the *exterior* of  $\mathcal{W}_i^{q_j}$  to its *boundary*  $((\mathcal{W}_i^{q_j})^c \text{ to } \partial\mathcal{W}_i^{q_j})$ .

Recall the hybrid procedural model  $H_{\text{procedure}}$  of the yellow interval problem (Figure 13). The discrete abstraction  $G_{\text{procedure}}^*$  has more than  $3^2 = 9$  modes since we separate not only  $\mathcal{W}_{\text{yellow}}$  and  $\mathcal{W}_{\text{red}}$ , but also the boundaries of  $\mathcal{W}_{\text{green}}$ ,  $\mathcal{W}_{\text{yellow}}$ , and  $\mathcal{W}_{\text{red}}$ , according to the type of action the driver must follow. Table 2 indicates the correspondence between  $G_{\text{procedure}}^*$  of Figure 24 to the state-space analysis in Figures 3 and 4 by noting the following correspondences: Note that  $\mathcal{W}_{\text{yellow}} = \mathcal{W}_{\text{accel}} \cup \mathcal{W}_{\text{coast}} \cup \mathcal{W}_{\text{brake}}$  and  $\mathcal{W}_{\text{red}} = \mathcal{W}_{\text{brake}} \cup \{[L, \infty) \times (0, v_{\text{max}}]\}$ . Also note that  $\partial\mathcal{W}_{\text{brake}}$  is defined for  $v > 0$ ; and that the point  $(x, v) = (0, 0)$  is denoted by the indication STOP.

### 3.5 Use of the Discrete Abstraction

The discrete abstraction  $G_{\text{procedure}}^*$  of the hybrid human-automation system  $H_{\text{procedure}}$  can now be used to analyze, verify, or design user-interfaces for  $H_{\text{procedure}}$ . A user-interface can be modeled by a discrete event system, with the modes determined by the indications on the display, and events determined by internal transitions in the system, or by the user's actions. The user activates various knobs, buttons, and toggles to change the hybrid system's mode. The interaction between the user's

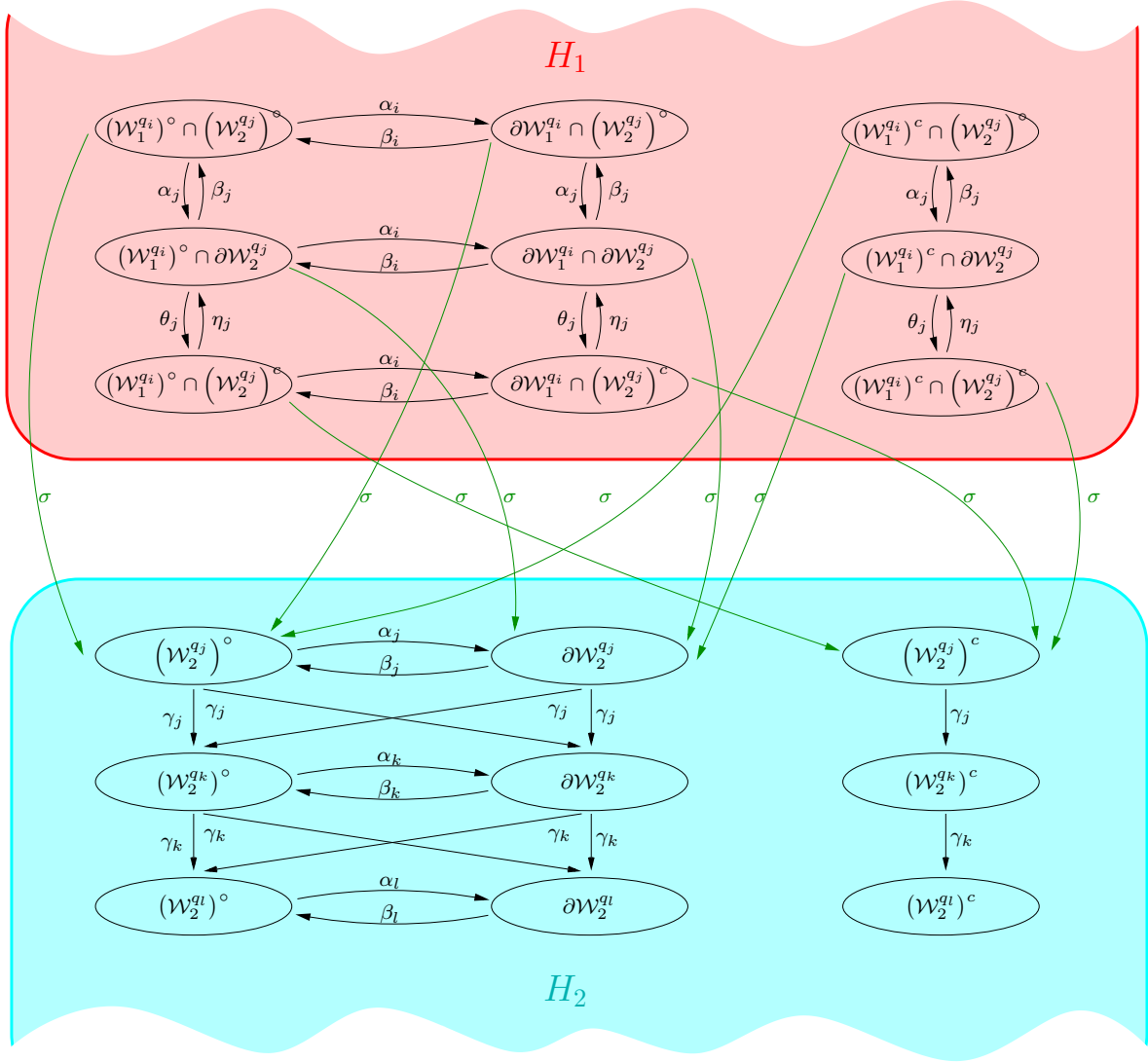


Figure 23: Portion of discrete model  $G_{\text{procedure}}^*$  across hybrid subsystems  $H_1$  and  $H_2$ . Modes before the user-controlled switch  $\sigma$  are modeled deterministically; the rest of the hybrid subsystems are modeled nondeterministically to reduce the complexity of the abstracted model.

Light Color	Region	Mode in $G^{\text{procedure}}$ (Figure 24)
Green (Figure 4)	shaded	$\mathcal{W}_{\text{green}}^{\circ}$
	braking boundary	$\partial\mathcal{W}_{\text{green-brake}}$
	accelerating boundary	$\partial\mathcal{W}_{\text{green-accel}}$
Yellow (Figure 3)	unshaded	$\mathcal{W}_{\text{green}}^c$
	(1), (4), (5)	$\mathcal{W}_{\text{brake}}^{\circ}$
	boundary of (1) and (4)	$\partial\mathcal{W}_{\text{brake}}$
	(2),(3),(4),(5),(6)	$\mathcal{W}_{\text{accel}}$
	(2),(4),(6)	$\mathcal{W}_{\text{coast}}$
	(7)	$\mathcal{W}_{\text{yellow}}^c$
	$(x, v) = (0, 0)$	STOP
Red (Figure 3)	(1), (4), (5)	$\mathcal{W}_{\text{brake}}^{\circ}$
	boundary of (1) and (4)	$\partial\mathcal{W}_{\text{brake}}$
	(6)	$\mathcal{W}_{\text{red}} \setminus \mathcal{W}_{\text{brake}}$
	(2), (3), (7)	$\mathcal{W}_{\text{red}}^c$
	$(x, v) = (0, 0)$	STOP

Table 2: Summary of abstraction from continuous state-space regions (Figures 3 and 4) to discrete modes of  $G^*_{\text{procedure}}$  (Figure 24).

actions and the hybrid system’s modes are encapsulated by a finite-state machine representation.

In Section 2, an interface was constructed based on the result of the reachability analysis. However, in some situations, an interface may already exist. In this case, it is of high interest to be able to verify that the interface correctly represents the underlying, hybrid system. The authors in [11, 25] developed a method to formally verify that one discrete system (such as an interface) is a correct abstraction of another discrete system (such as a “truth” or procedural model of a system). Using these methods, the existing interface interface can be verified against the discrete abstraction developed in Section 3.4, according to the method developed in [11, 25].

The same authors have also developed an interface design technique [1] using state reduction techniques for incompletely-specified finite state machines. State reduction involves creating a reduced finite state machine, based on a given output for a larger, more complex finite state machine. Presuming that the original, complex finite state machine represents the “truth” model, the reduced model represents the interface for the original system. While the interface in Section 2, constructed by hand from the reachability analysis, has only four modes, in general, the resulting discrete automaton may contain far more modes. These modes may represent more information than the user needs in order to accurately interact with the system. In this case, a correct and succinct interface can be constructed according to discrete-state reduction techniques as in [1].

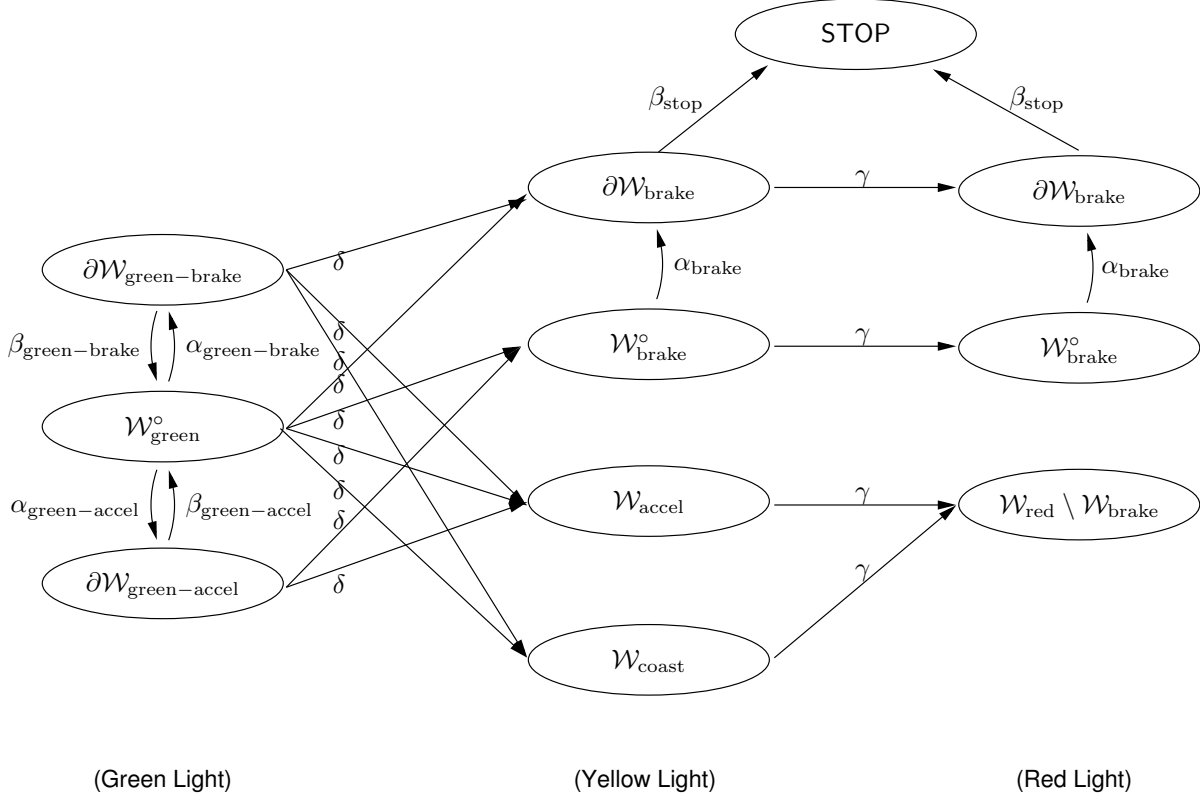


Figure 24: Discrete procedural model  $G_{\text{procedure}}^*$  for yellow interval problem. The modes in the set  $Q_{\text{procedure}}^*$  are defined in Table 2, and the set of events is  $\Sigma_{\text{procedure}}^* = \{\delta, \gamma\}$ . The event  $\delta$  occurs when the yellow light appears, and the event  $\gamma$  occurs  $\tau$  second later, when the red light appears.

## 4 Automatic Landing of a Commercial Aircraft

We now apply the method detailed in Section 3 to a more complex example: the automatic landing of a large civil jet aircraft. Automatic landing systems, as the name implies, use information from the aircraft as well as from the airport facilities to guide the aircraft to a smooth touchdown on the runway. While highly automated, autoland systems require pilot interaction and supervision in order to successfully complete a landing. Additionally, in the case of an aborted landing, the pilot intervenes to initiate an automated go-around maneuver. While the automation enacts the low-level control necessary for an aircraft to track desired landing and go-around trajectories, the pilot sets parameters, changes the aircraft's configuration, and enacts high-level mode changes.

Autoland systems are complex, safety-critical systems, and are subject to stringent certification criteria [34]. Modeling the aircraft's behavior, which incorporates logic from the autopilot as well as inherently complicated aircraft dynamics, results in a high-dimensional hybrid system with many continuous and discrete states. Naturally, only a subset of this information is displayed to the pilot. Choosing which information should be included in the subset is critical: this information must

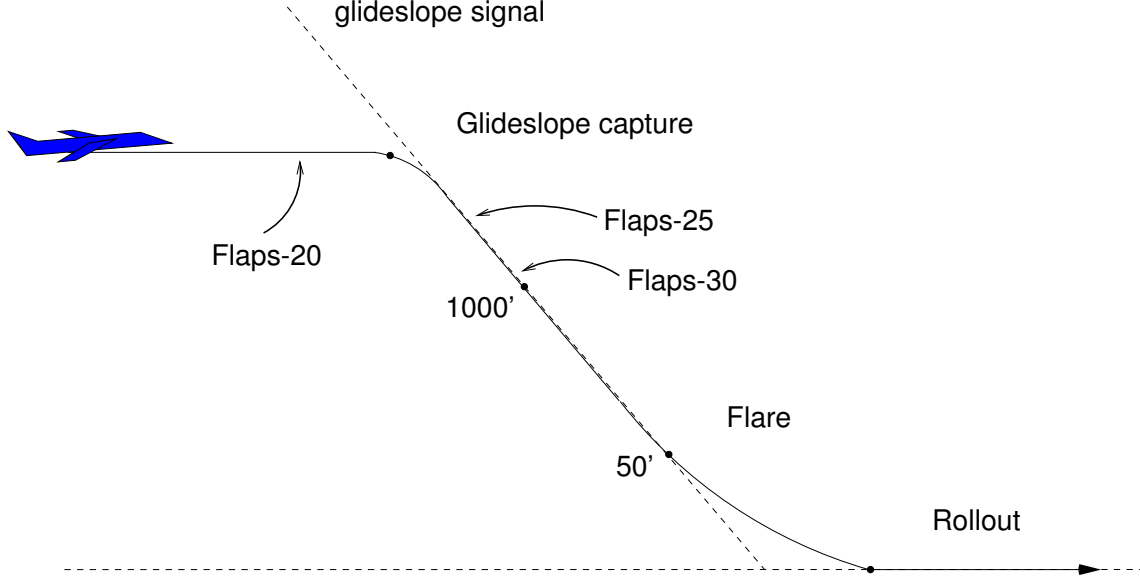


Figure 25: Typical landing scenario. The dots indicate transitions in autopilot mode during a typical landing procedure: ‘Approach’, ‘Capture’, ‘Glideslope’, ‘Flare’, and ‘Rollout’. Additionally, the pilot is instructed that the aircraft should be in Flaps-20 (with the landing gear down) before glideslope capture; and that the aircraft should sequence through Flaps-25 and Flaps-30 by the time the aircraft reaches 1000 feet.

adequately represent the hybrid system’s behavior. The problem of determining which information adequately represents the underlying hybrid system is closely related to problems in observability: a system is *observable* if the state of the system can be accurately reconstructed from the output of the system. The output often represents a reduced set of information about the state of the system. While both continuous and discrete system observability are well-researched issues [35], hybrid system observability is a more recent topic, with results mainly for hybrid systems with linear or affine continuous dynamics [36, 37, 38, 39].

We wish to determine, for an automatic landing/go-around maneuver of a highly automated civil jet aircraft, which information the pilot must have in order to safely maneuver the aircraft. We will first discuss the autoland/go-around maneuver in more detail, then formulate it as a hybrid human-automation system in which it is critical that the continuous state remain within certain bounds to maintain safe operation. We separate the hybrid system into two hybrid subsystems, and perform a hybrid reachability analysis on each subsystems. As in detailed Section 3.4, the reachability results provide the means to create a discrete system which represents the hybrid human-automation system. This abstraction can then be used in user-interface analysis and design.

In a typical autoland maneuver (Figure 25), the aircraft begins its approach approximately 10 nautical miles from the touchdown point. The pilot extends the landing gear (down) sets the flaps, aerodynamic surfaces on the trailing edge of the wings, to the setting Flaps-20. The aircraft descends

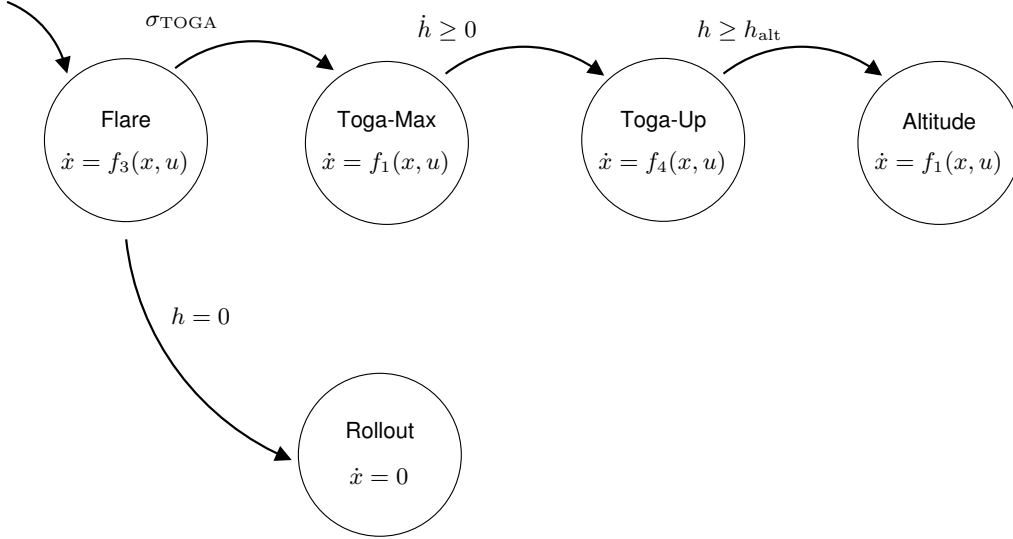


Figure 26: Hybrid procedural automaton  $H_{\text{procedure}}$ . The dynamics are described in Section 4.1 and correspond to aerodynamics constants enumerated in Table 3.

towards the glideslope, an inertial radio beam which the aircraft can track. The autopilot captures the glideslope signal about five nautical miles from the touchdown point. The pilot extends the flaps, which increases drag on the wing and maintains lift at low speeds. The pilot extends the flaps through Flaps-25 and Flaps-30 before the aircraft reaches 1000' altitude. At approximately 50', the aircraft leaves the glideslope and begins the flare maneuver, which allows the aircraft to touchdown smoothly on the runway with an appropriate descent rate.

If for any reason the pilot or air traffic controller deems the landing unacceptable (debris on the runway, a potential conflict with another aircraft, or severe wind shear near the runway, for example), the pilot must initiate a go-around maneuver. A go-around maneuver can be initiated at any time after the glideslope has been captured and before the aircraft touches down. In a typical autoland, pushing the go-around button engages a sequence of events designed to make the aircraft climb as quickly as possible to a preset missed-approach altitude (usually about 2500').

#### 4.1 Hybrid Procedural Automaton

The hybrid procedural model (Figure 26) focuses on a small portion of the autoland procedure, beginning with the flare maneuver. We consider the longitudinal dynamics only, in which the continuous state  $x = [V, \gamma, h] \in \mathbb{R}^3$  represents the aircraft's speed  $V$ , flightpath angle  $\gamma$ , and altitude  $h$ . The continuous inputs are  $u = [T, \alpha]$ , assuming direct control over the aircraft's thrust  $T$  and angle of attack  $\alpha$ .

The initial state of the procedural model is the Flare mode (Figure 26). In Flare mode, the

flaps are at Flaps-30 and the thrust is fixed at idle. During normal autolands, the aircraft switches to Rollout mode upon touchdown. (We do not model the aircraft’s behavior as it rolls along the runway.) When a pilot initiates a go-around maneuver (often called a “TOGA” due to the “Take-Off/Go-Around” indicator on the pilot display), the pilot changes the flaps to Flaps-20 and the autothrottle forces the thrust to  $T_{\max}$  (**Toga-Max** mode). When the aircraft obtains a positive rate of climb, the pilot raises the landing gear and the autothrottle allows the thrust to take on any value in the range  $T \in [0, T_{\max}]$  (**Toga-Up** mode). The aircraft continues to climb to the missed approach altitude  $h_{\text{alt}}$ , and then automatically switches into an altitude-holding mode, **Altitude**.

Go-arounds are unpredictable and may be required at any time during the autoland prior to touchdown. Yet, only the pilot has the authority to initiate this maneuver. We therefore model  $\sigma_{\text{TOGA}}$  as a user-controlled transition. We model certain events as simultaneous:  $\sigma_{\text{TOGA}}$  and changing the flaps to Flaps-20, and  $\dot{h} \geq 0$  and raising the landing gear.

As in [40], we model the nonlinear longitudinal dynamics in the path frame as

$$\begin{bmatrix} m\dot{V} \\ mV\dot{\gamma} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} -D(\alpha, V) + T \cos \alpha - mg \sin \gamma \\ L(\alpha, V) + T \sin \alpha - mg \cos \gamma \\ V \sin \gamma \end{bmatrix} \quad (9)$$

We assume the aircraft has mass  $m = 190000$  kg, and the gravitational constant is  $g = 9.81$  m/s<sup>2</sup>. The aircraft has pitch  $\theta = \alpha + \gamma$ . In equation (9), lift  $L(\alpha, V)$  and drag  $D(\alpha, V)$  account for the specific aerodynamic properties of the aircraft of interest. They can be obtained from the dimensionless lift and drag coefficients  $C_L$  and  $C_D$ ,

$$L(\alpha, V) = \frac{1}{2} \rho V^2 S C_L(\alpha) \quad (10)$$

$$D(\alpha, V) = \frac{1}{2} \rho V^2 S C_D(\alpha) \quad (11)$$

with air density  $\rho = 1.225$  kg/m<sup>3</sup> and wing surface area  $S = 427.80$  m<sup>2</sup>.

$$C_L(\alpha) = C_{L_0} + C_{L_\alpha} \alpha \quad (12)$$

$$C_D(\alpha) = C_{D_0} + K C_L^2(\alpha) \quad (13)$$

Equation (12) results from thin airfoil theory [41], and can be adapted to a full aircraft [42] by an appropriate choice of the positive constants  $C_{L_0}$  and  $C_{L_\alpha}$ . Equation (13), known as the “drag polar”, results from lifting line theory [41] and applies to a clean wing (no fuselage and engines), but can be extended to a full aircraft [42], by incorporating aircraft characteristics in the coefficients  $C_{D_0}$  and  $K$ . The coefficients  $C_{L_0}$ ,  $C_{L_\alpha}$ ,  $C_{D_0}$  and  $K$  thus depend on the flight configuration of

Dynamics	$C_{L_0}$	$C_{D_0}$	$K$	Flaps Setting	Landing Gear
$\dot{x} = f_1(x, u)$	0.4225	0.024847	0.04831	Flaps-20	Down
$\dot{x} = f_2(x, u)$	0.7043	0.025151	0.04831	Flaps-25	Down
$\dot{x} = f_3(x, u)$	0.8212	0.025455	0.04831	Flaps-30	Down
$\dot{x} = f_4(x, u)$	0.4225	0.019704	0.04589	Flaps-20	Up
$\dot{x} = f_5(x, u)$	0.7043	0.020009	0.04589	Flaps-25	Up
$\dot{x} = f_6(x, u)$	0.8212	0.020313	0.04589	Flaps-30	Up

Table 3: Aerodynamic constants for autoland modes.

Mode	$V$ [m/s]	$\gamma$ [degrees]	$\alpha$ [degrees]	$T$ [N]
Flare	[55.57, 87.46]	$[-6.0^\circ, 0.0^\circ]$	$[-9^\circ, 15^\circ]$	0
Toga-Max	[63.79, 97.74]	$[-6.0^\circ, 0.0^\circ]$	$[-8^\circ, 12^\circ]$	$T_{\max}$
Toga-Up	[63.79, 97.74]	$[0.0^\circ, 13.3^\circ]$	$[-8^\circ, 12^\circ]$	$[0, T_{\max}]$
Altitude	[63.79, 97.74]	$[-0.7^\circ, 0.7^\circ]$	$[-8^\circ, 12^\circ]$	$[0, T_{\max}]$

Table 4: State bounds for autoland modes of  $H_{\text{procedure}}$ .

the aircraft as well as its geometry. We calculated and estimated these values, which are typical for large civil aircraft, from data in [40, 43, 44, 45, 46]. The lift coefficient slope  $C_{L_\alpha} = 5.105$  for all modes; other coefficients are summarized in Table 3. The computation of  $C_{L_0}$  includes estimates of the additional lift provided by flaps and slats. The constant  $C_{D_0}$  includes terms due to aircraft aerodynamic shape, deployment of landing gear, flap and slat deflection. The constant  $K$  is determined analytically [41] as a function of the aerodynamic characteristics of the aircraft: aspect ratio, which is the ratio of the span length squared to the wing surface area, and span efficiency factor, which relates the lift of a given wing to an ideal, elliptical lift distribution.

Each mode in the procedural automaton is subject to state and input bounds, due to constraints arising from aircraft aerodynamics and desired aircraft behavior. These are summarized in Table 4. The bounds on  $V$  and  $\alpha$  are determined by stall speeds and structural limitations for each flap setting [46, 42]. The stall angle (maximum value of  $\alpha$ ) is the value of  $\alpha$  which corresponds to the maximum lift coefficient. The maximum allowed  $\alpha$  is thus determined for each aircraft using this value. The stall speed can be computed analytically, using the landing configuration of the aircraft (mass, wing surface area, maximum lift coefficient, air density). Bounds on  $\gamma$  and  $T$  are determined by the desired maneuver [47, 48], with  $T_{\max} = 686700$  N. Additionally, at touchdown,  $\theta \in [0^\circ, 12.9^\circ]$  to prevent a tail strike, and  $\dot{h} \geq -1.829$  m/s to prevent damage to the landing gear. These restrictions determine the allowable regions of operation in landing and go-around maneuvers.

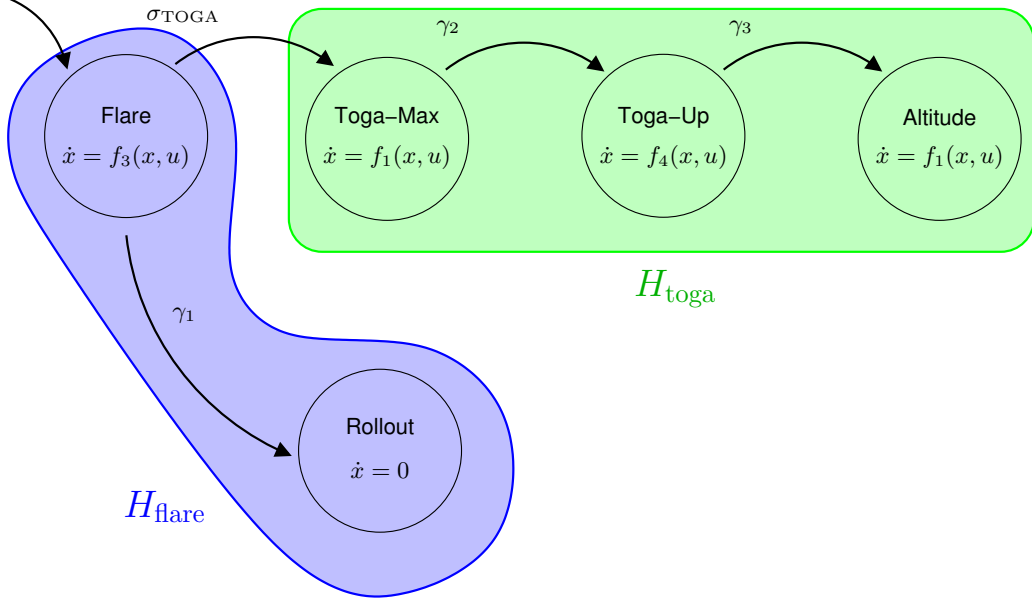


Figure 27: Hybrid subsystems for landing/go-around scenario. The events are relabeled from  $H_{\text{procedure}}$  as shown in Figure 26, so that  $\gamma_1$  occurs when  $h = 0$ ,  $\gamma_2$  occurs when  $\dot{h} \geq 0$ , and  $\gamma_3$  occurs when  $h \geq h_{\text{alt}}$ .

Subsystem	$V$ [m/s]	$\gamma$ [degrees]	$h$ [m]
$H_{\text{flare}}$	[50, 100]	[-8, 17]	[-5, 20]
$H_{\text{toga}}$	[50, 100]	[-8, 17]	[-5, 20]

Table 5: Computational domain for autoland hybrid subsystems.

## 4.2 Reachability Analysis of Hybrid Subsystems

In order to determine the subset of the allowable region of operation in which we can guarantee the aircraft can remain safe, we separate the hybrid procedural model  $H_{\text{procedure}}$  across the user-controlled switch  $\sigma_{\text{TOGA}}$ . This separation results in two hybrid subsystems,  $H_{\text{flare}}$  and  $H_{\text{toga}}$ , as shown in Figure 27. We complete a hybrid reachability analysis within each hybrid subsystem in order to determine the set of continuous states in each subsystem, in which the aircraft can land or go-around, respectively, while meeting the state and input constraints detailed in Section 4.1. The constraints for the aircraft to land define  $(\mathcal{W}_{\text{flare}})_0$ , and the constraints for the aircraft to go-around define  $(\mathcal{W}_{\text{toga}})_0$ . In the computation of the reachable set for each subsystem, automatic transitions are smoothly accomplished in by modeling the change in dynamics across the switching surface as another nonlinearity in the dynamics. Additionally, for the reachability computation we assume in  $H_{\text{toga}}$  that if the aircraft leaves the top of the computational domain ( $h = 20$  m) without exceeding its flight envelope, it is capable of achieving **Altitude** mode, which we consider safe. The computational domain for each subsystem is indicated in Table 5.

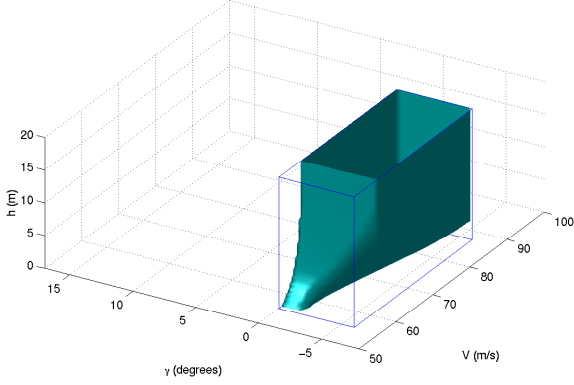


Figure 28: Allowable region  $(\mathcal{W}_{\text{flare}})_0$  (states inside wireframe box) and safe region  $\mathcal{W}_{\text{flare}}$  (states within solid) in hybrid subsystem  $H_{\text{flare}}$ .

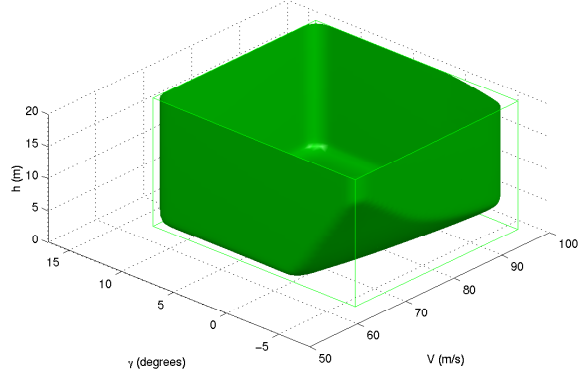


Figure 29: Allowable region  $(\mathcal{W}_{\text{toga}})_0$  (states within wireframe box) and safe region  $\mathcal{W}_{\text{toga}}$  (states within solid) in hybrid subsystem  $H_{\text{toga}}$ .

Figure 28 depicts the boundary of the allowable region  $(\mathcal{W}_{\text{flare}})_0$  (shown as the wireframe box) as well as the computational result for  $\mathcal{W}_{\text{flare}}$  (solid). Similarly, Figure 29 depicts the boundary of  $(\mathcal{W}_{\text{toga}})_0$  (wireframe box) as well as  $\mathcal{W}_{\text{toga}}$  (solid). Figure 30 shows the intersection of the two safe regions in the state-space. The solid shape indicates the region of intersection between the two modes: in this region both safe landing and safe go-around are possible. The analysis shows that when the aircraft begins in Flare, there are states from which a safe landing is possible, but a safe go-around is not.

### 4.3 Discrete Abstraction

It is now possible to use the result of the hybrid reachability analysis to abstract from it a discrete system. Such a discrete system, which encompasses the regions calculated through the hybrid reachability analysis, can then be used to perform further analysis, such as interface analysis, verification, or design.

In most commercial aircraft, the low-level control is performed by the autopilot, which has authority over small control surface movement. The details of the low-level control are hidden from the pilot, who anticipates system behavior by understanding the behavior of each autopilot mode. In some instances, the pilot allows the autopilot to directly control mode switches. We therefore assume an automated controller enforces  $u = u^*(x)$ , but leave it to the pilot to enforce any discrete switches necessary to maintain safety. This assumption mimics the hierarchical nature of highly automated aircraft, in which pilots supervise the automation by enacting discrete switches. In this supervisory role, the pilots always have the option *not* to enact a recommended switch.

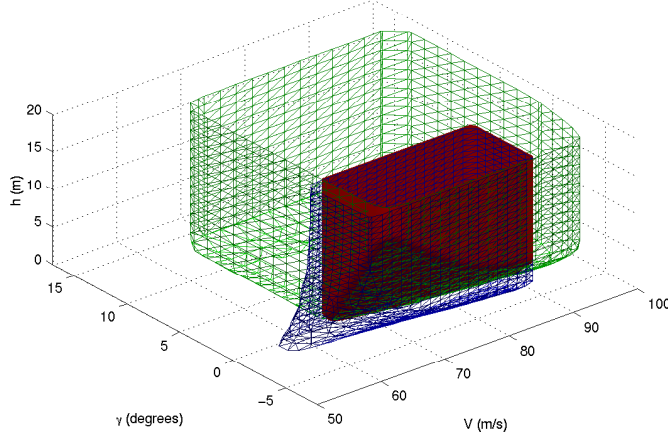


Figure 30: The solid (red) shape is the safe region  $\mathcal{W}_{\text{flare}} \cap \mathcal{W}_{\text{toga}}$ , from which safe landing and safe go-around is possible. The meshes depict  $\mathcal{W}_{\text{flare}}$  (dark mesh, dark blue) and  $\mathcal{W}_{\text{toga}}$  (light mesh, green).

There is only one mode in  $H_{\text{procedure}}$  from which a user-controlled transition ( $\sigma_{\text{TOGA}}$ ) is possible: Flare. This mode is abstracted according to its partition with  $\text{Toga-Max} = R_{\text{procedure}}(\text{Flare}, \sigma_{\text{TOGA}})$ , so results in  $3^{(1+1)} = 9$  modes in  $G_{\text{procedure}}^*$ , as shown in Figure 31. The remaining modes (Rollout, Toga-Max, Toga-Up, and Altitude) each result in  $3^{(1+0)} = 3$  discrete modes in  $G_{\text{procedure}}^*$ , since no user-controlled transitions are possible from these modes in  $H_{\text{procedure}}$ . The modes in the upper half of Figure 31 correspond to modes in  $H_{\text{flare}}$ , while modes in the lower half correspond to  $H_{\text{toga}}$ . For clarity, the modes which correspond to safe regions of the state-space are solidly colored; modes which correspond to unsafe regions have hashed marks. While Figure 31 depicts the entire state-space of  $H_{\text{procedure}}$ , we restrict the initial states  $Q_{\text{procedure}}^0$  of  $G_{\text{procedure}}^*$  to lie within or on the boundary of the region for safe landing in Flare mode.

#### 4.4 Implications for User-Interface Analysis

The results of the hybrid reachability analysis show that there exists a region from which safe landing is assured, but a safe go-around is not assured. This region, of course, is problematic. When we consider possible pilot interaction with the model of the system described here, there are two additional problems: 1) the pilot is unaware of this region, and 2) even if the pilot is told about the existence of this region, there is no display in the cockpit to tell the pilot that the autopilot is in a region from which a safe go around is not assured.

It is possible, however, to provide feedback to the pilot about this region (from which a safe go-around is not assured). Using the same approach that we discussed earlier in the automotive problem, it is possible to develop an interface that will indicate to the pilot when the aircraft is about to enter the region from which a safe go around is not assured. Therefore, one of the

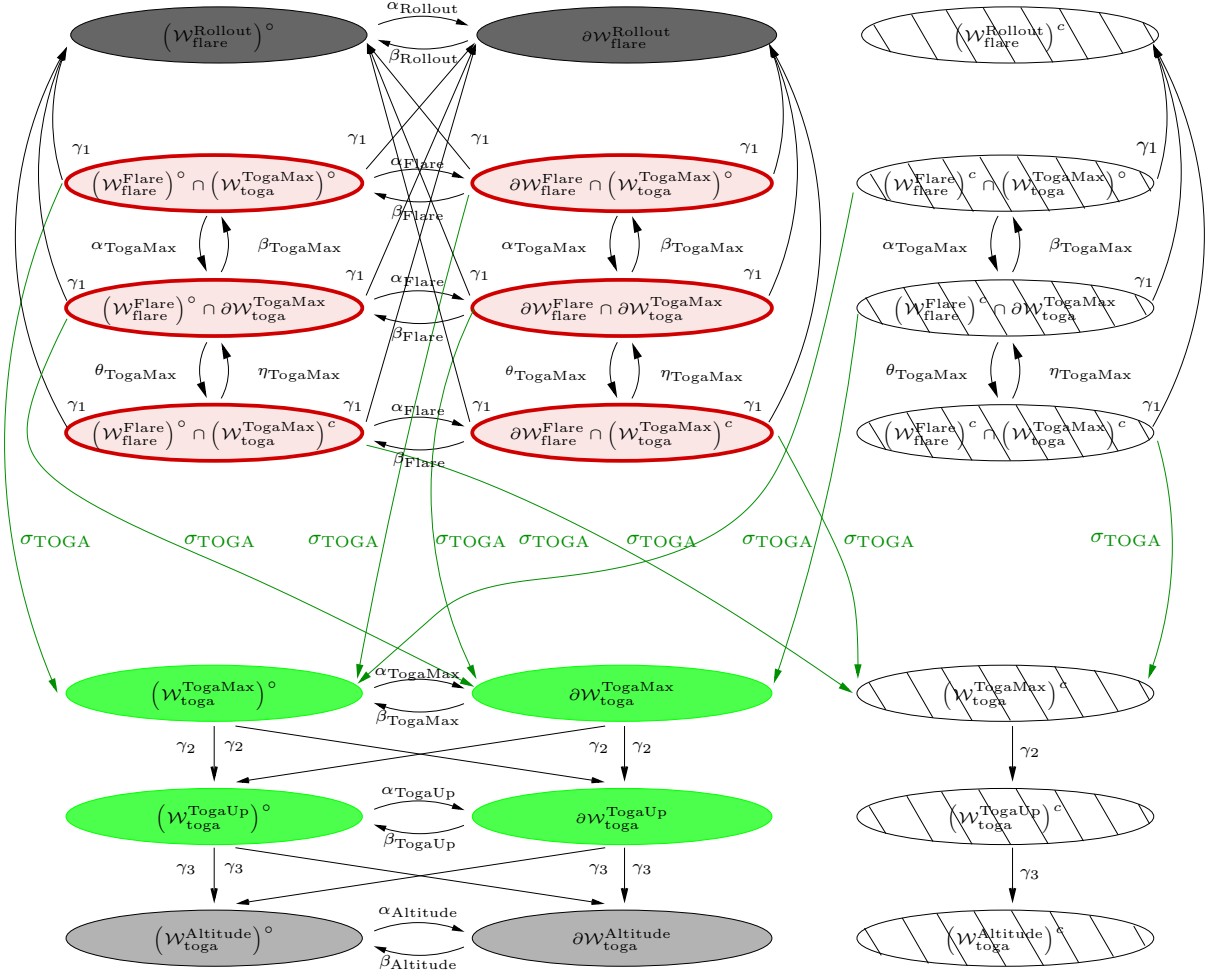


Figure 31: Abstracted procedural model  $G^*_{\text{procedure}}$  for autoland/go-around maneuver. Modes in the set of initial modes  $Q^0_{\text{procedure}}$  are bolded.

pilot's tasks, in this case, with respect to the model discussed above, is to avoid entering the unsafe go-around region. The design of such an (advisory) interface must take into account the pilot's reaction time, the dynamics of the airplane, and the various control options (such as increased thrust) that are available to the pilot to prevent the airplane from entering the problematic region.

There are many different ways to inform the pilot about possible actions to keep the aircraft in a safe region of operation. One way to advise the user about this situation is to create an interface which differentiates between the region of the state-space in **Flare** from which a safe go-around is possible, and the region in **Flare** from which a safe go-around is not possible. The interface could also indicate possible continuous control actions, such as "Increase speed" or "Climb", that would help the pilot reach the region of **Flare** from which a safe go-around is possible. Alternatively, the restrictions this procedure assumed (such as concurrent transitions) can be relaxed to allow the pilot more control actions as well as increase the region in **Flare** from which a go-around can

be safely accomplished. The hybrid reachability analysis and subsequent abstraction to a discrete system would be performed by the same method as presented here.

However, if even after these steps, there are still regions from which a safe go-around is not possible according to the maneuvers specified, we could complete a separate reachability analysis to determine regions in the state-space from which a recovery to safety is possible. As opposed to the computation of the unsafe region during the green light in the yellow interval problem (Figure 5), this computation would involve specifying *additional* control actions that the pilot or the aircraft could take, as well as a new formulation of the reachability problem. The computed results of the analysis already presented (Figure 30) would be used as the initial conditions in a new reachability analysis: this analysis would tell us whether it is possible, with these additional control actions, to reach the safe region of operation from the states which, according the prior analysis and prior control actions, a safe go-around is not possible. This differs from the prior analysis in that we now wish to determine those set of states which can, through a recovery maneuver, reach the safe regions of Figure 30.

These approaches are directions for future work.

## 5 Implications for User-Interface Design and Analysis

This report presents a methodology for the analysis of automated hybrid control systems that include user interaction. The methodology makes use of hybrid reachability algorithms and computational tools [18, 19]. The advantage of this methodology is that it incorporates, by construction, the coupling between the continuous behaviors of the dynamical system, discrete mode logic, as well as user interaction with the system. As such, the methodology presented here has two important implications for user interface design.

First, as shown in the automotive example and in the case of the autoland example, it is possible to use this hybrid reachability methodology to analyze and identify different regions within the operational state-space of the system. These regions are important to the user, because they prompt decisions and/or control actions. At the very least, it is important that the user know about and understand the different regions in the operating space that affect the user’s interaction and supervision of the system. This is critical for designing interfaces for complex automated systems, and to this end, the hybrid reachability analysis can be used as the foundation for the construction and design the user-interface.

With regards to interface design, the approach presented here is not concerned with the layout and color-coding of the interface, but rather in the information content provided on the interface. Namely, we are interested in determining what information the user needs in order to control the system safely. Nevertheless, just indicating to the user in what region of the state space the system is in – is not enough. When it comes to unsafe regions, the user must also be provided with advance information (warning) when the system is about to enter the unsafe region. As shown in the automotive example, there are various control schemes and corresponding interface indications that can be employed to aid the driver in the task of avoiding the unsafe region.

The second important implication of this work is for verification of interfaces. The methodology described here shows how, with the aid of the hybrid system reachability tool, it is possible to transform a complex hybrid system to an equivalent discrete representation. This (abstracted) representation is very important for interface design, because it makes it possible application of existing interface verification techniques. Specifically, once the hybrid system is described in a discrete representation and the interface is modeled as a discrete event system, it is possible to verify that an interface is correct. Namely, that there exist no error modes, no blocking modes, and no augmenting modes [1, 25].

An error mode represents a divergence between the underlying system and the user interface. For example, if the system is in one mode but the interface shows that the system is in another mode, the system has a serious design flaw. A blocking mode represents a situation in which the

user is unaware that certain events can trigger mode changes in other parts of the system. The interface abstracts away this information, but if the user accidentally or unintentionally presses the button at the wrong time, the system will switch modes (to user's dismay). An augmenting mode represents a situation in which the interface indicates that a certain mode change is possible, when in fact the machine has no such mode or it can not enter it (for example, if it is disabled). (See [25] for additional information regarding these interfaces verification criteria and the mechanics of the interface verification process).

The existence of error, blocking, or an augmenting modes in any interface is a serious design deficiency that should be avoided. This is particularly acute when considering safety-critical systems, and this is why interface verification is critical. The methodology described here, coupled with the techniques described in [25], are an important step toward verification of user interaction with hybrid systems.

## 6 Conclusions and Limitations

There is an ongoing debate in aviation, space, and other safety-critical industries about the role of the operator and the extent to which automation can and should be used [3, 4]. This debate has been fueled by incidents and accidents in which operators were surprised about the behavior of the automation. While the debate will continue for decades to come, it is clear that some of the problems in human-automation interaction stem from design problems. The methodology described in this report aims to identify these design problems before they contribute to mishaps.

This report presents a methodology for analysis of hybrid systems which encompass user-interaction. The formal methodology discussed here involves three steps: 1) separation of the hybrid procedural model into hybrid subsystems across user-controlled switches, 2) hybrid reachability analysis and controller synthesis of each hybrid subsystem, and 3) abstraction to a discrete system based on the reachability result. Once the hybrid system is described as an equivalent discrete system, and the user-interface is also modeled as a discrete system, it is possible to make an additional step and use the verification methods of [25] to ensure that the interface is correct.

Two examples were presented in this report: the yellow interval dilemma, and pilot interaction with aircraft automation during a go-around maneuver. These examples served to demonstrate the methodology and its application. Because it is a general methodology, it is applicable to more complicated systems and more elaborate descriptions of procedures and interfaces than those presented here. The key contribution of this work is to create an abstraction of hybrid systems based on a hybrid reachability result: the resultant discrete system is one for which existing interface verification and design techniques [1, 25] can be implemented.

The methodology presented here makes assumptions regarding control theoretic, human-factors, and operational issues. The main control theoretic assumptions involve hierarchical control as well as complete and accurate state measurements. Implementation of hierarchical goals and control schemes is an active topic of research [49, 50, 51]. We assume that the low-level controllers which enforce safety are already in place and work flawlessly with other goals. Smoothly incorporating potentially conflicting goals is a difficult problem. In highly automated aircraft, envelope protection schemes are frequently added on to existing control schemes [48], and validated in simulators. Additionally, as stated in Section 3, we assume full and accurate information about the continuous state. Although the controllers for safety will likely have full access to the continuous state, in any real system there will inevitably be noise in state measurements, and estimates of the state will be necessary. A clear direction of future work is to examine this same verification problem in the context of uncertain state measurements.

With regards to human factors issues, the methodology presented here only addresses *what*

information is displayed, since this can be quantified within a known mathematical framework. It does not address how the information is displayed.

With regards to operational issues, the methodology presented here assumes a specified *procedure*. Procedures are described in manuals and taught to operators (such as pilots) during training. To be able to verify user-interaction with hybrid systems, we assumed in this report that the user strictly follows a given procedure. However, operators do not always follow a procedure [52]. Although specific sequences of user actions must be encoded in the procedural model, this model can always be extended to incorporate additional sequences of user actions, even all possible sequences of the “truth” model

While we have focused in this section on the limitations of our assumptions, it is important to note that the main advantage of this methodology is that it incorporates, by construction, the coupling between the continuous behaviors of the aircraft and the automation and autopilot logic. Verification within a hybrid framework accounts for the inherently complicated dynamics underlying the simple, discrete representations displayed to the user. The method presented here directly accounts for the user’s prerogative in determining system safety in modeling of the user’s interaction with the system. While no guarantees can be made about the user’s actions, this methodology provides a mathematical guarantee that the discrete abstraction contains correct information regarding the effect of the user’s actions on system safety. With this abstraction, an interface can be designed to ensure that user is provided with the correct information to complete the desired procedure or task.

## Acknowledgments

We would like to acknowledge Michael Heymann for his contributions to the interface analysis and verification methods which inspired this work. We would also like to thank David Austin, Randall Mumaw, and Charles Hynes for their help regarding the aircraft autoland scenario. Ian Mitchell developed the computational tool used to obtain the hybrid reachability result, and Alexandre Bayen developed a generic civil jet aircraft model method which was adapted for our use in the aircraft autoland scenario.

## References

- [1] M. Heymann and A. Degani, “On abstractions and simplifications in the design of human-automation interfaces,” NASA Technical Memorandum 211397, NASA Ames Research Center, Moffett Field, CA, 2002.
- [2] Federal Aviation Administration, “Federal aviation regulations,” FAR 121.1329, 2002.
- [3] C. Billings, *Aviation Automation: The Search for a Human-Centered Approach*. Hillsdale, NJ: Erlbaum, 1997.
- [4] E. Wiener and R. Curry, “Flight-deck automation: promises and problems,” NASA Technical Memorandum 81206, NASA Ames Research Center, Moffett Field, CA, June 1980.
- [5] R. Parasuraman, T. Sheridan, and C. Wickens, “A model for types and levels of human interaction with automation,” *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, vol. 30, May 2000.
- [6] N. Sarter, D. Woods, and C. Billings, “Automation surprises,” in *Handbook of Human Factors and Ergonomics*, pp. 1295–1327, NY: John Wiley and Sons, Inc., 1999.
- [7] N. Sarter and D. Woods, “How in the world did we get into that mode? Mode error and awareness in supervisory control,” *Human Factors*, vol. 37, no. 1, pp. 5–19, 1995.
- [8] E. Palmer, “Oops, it didn’t arm - a case study of two automation surprises,” in *8th International Symposium on Aviation Psychology*, (Columbus, Ohio), 1995.
- [9] J. Rushby, “Analyzing cockpit interfaces using formal methods,” in *Electronic Notes in Theoretical Computer Science*, 43, Elsevier Science B.V., 2001.
- [10] R. Butler, S. Miller, J. Potts, and V. Carreno, “A formal methods approach to the analysis of mode confusion,” in *Proceedings of the AIAA/IEEE Digital Avionics Systems Conference*, pp. C41/1–C41/8, 1998.
- [11] A. Degani, M. Heymann, G. Meyer, and M. Shafto, “Some formal aspects of human-automation interaction,” NASA Technical Memorandum 209600, NASA Ames Research Center, Moffett Field, CA, April 2000.
- [12] N. Leveson and E. Palmer, “Designing automation to reduce operator errors,” in *In the Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, (Orlando, FL), pp. 1144–1150, 1997.

- [13] N. Leveson, L. Pinnel, S. Sandys, S. Koga, and J. Reese, "Analyzing software specifications for mode confusion potential," in *Proceedings of a Workshop on Human Error and System Development* (C. Johnson, ed.), pp. 132–146, Glasgow, Scotland: Glasgow Accident Analysis Group, March 1997.
- [14] S. Vakil, A. Midkiff, T. Vaneck, and R. Hansman, "Mode awareness in advanced autoflight systems," in *Proceedings of the 6th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Man-Machine Systems*, (Cambridge, MA), 1995.
- [15] S. Vakil and J. Hansman, "Analysis of complexity evolution management and human performance issues in commercial aircraft automation systems," ICAT-2000-3, Massachusetts Institute of Technology, Cambridge, MA, May 2000.
- [16] M. Oishi, I. Mitchell, A. Bayen, C. Tomlin, and A. Degani, "Hybrid verification of an interface for an automatic landing," in *Proceedings of the IEEE Conference on Decision and Control*, (Las Vegas, NV), pp. 1607–1613, December 2002.
- [17] C. Tomlin, *Hybrid Control of Air Traffic Management Systems*. PhD thesis, University of California, Berkeley, CA, September 1998.
- [18] C. Tomlin, J. Lygeros, and S. Sastry, "A game theoretic approach to controller design for hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 949–970, 2000.
- [19] I. Mitchell, A. M. Bayen, and C. J. Tomlin, "Computing reachable sets for continuous dynamic games using level set methods," *IEEE Transactions on Automatic Control*. Submitted, December 2001.
- [20] I. Mitchell, *Application of Level Set Methods to Control and Reachability Problems in Continuous and Hybrid Systems*. PhD thesis, Department of Scientific Computing and Computational Mathematics, Stanford University, Stanford, CA, August 2002.
- [21] E. Clarke and R. Kurshan, "Computer-aided verification," *IEEE Spectrum*, vol. 33, pp. 61–67, June 1996.
- [22] Intel Corporation, "Detailed statistical analysis of floating point flaw in pentium processors." White Paper, November 1994. <http://support.intel.com/support/processors/pentium/fdiv/wp>.
- [23] J. Rushby, "Using model checking to help discover mode confusions and other automation surprises," in *Proceedings of the Workshop on Human Error, Safety, and System Development (HESSD)*, (Belgium), June 1999.

- [24] J. Crow, D. Javaux, and J. Rushby, “Models and mechanized methods that integrate human factors into automation design,” in *International Conference on Human-Computer Interaction in Aeronautics*, (Toulouse, France), September 2000.
- [25] A. Degani and M. Heymann, “Formal verification of human-automation interaction,” *Human Factors*, vol. 44, no. 1, pp. 28–43, 2002.
- [26] I. Mitchell, A. Bayen, and C. Tomlin, “Validating a Hamilton-Jacobi approximation to hybrid system reachable sets,” in *Hybrid Systems: Computation and Control* (M. D. Benedetto and A. Sangiovanni-Vincentelli, eds.), LNCS 2034, pp. 418–432, Springer Verlag, March 2001.
- [27] A. Crawford, “Driver judgment and error during the amber period at traffic lights,” *Ergonomics*, vol. 5, pp. 513–532, October 1962.
- [28] ITE Technical Council Task Force 4TF-1, “Determining vehicle signal change and clearance intervals,” tech. rep., Institute of Transportation Engineers, Washington, D.C., August 1994.
- [29] W. Stimpson, P. Zador, and P. Tarnoff, “The influence of the time duration of yellow traffic signals on driver response,” *ITE Journal*, pp. 22–29, November 1980.
- [30] C. Liu, R. Herman, and D. Gazis, “A review of the yellow interval dilemma,” *Transportation Research A*, vol. 30, no. 5, pp. 333–348, 1996.
- [31] D. Gazis, R. Herman, and A. Maradudin, “The problem of the amber signal light in traffic flow,” *Operations Research*, vol. 8, pp. 112–132, January-February 1960.
- [32] Office of the Majority Leader, U.S. House of Representatives, “The red light running crisis: Is it intentional?,” May 2001. <http://www.freedom.gov/auto>.
- [33] I. Mitchell and C. Tomlin, “Level set methods for computation in hybrid systems,” in *Hybrid Systems: Computation and Control* (B. Krogh and N. Lynch, eds.), LNCS 1790, Springer Verlag, March 2000.
- [34] Federal Aviation Administration, “Criteria for approval of Category III weather minima for takeoff, landing, and rollout,” Advisory Circular 120-28D, U.S. Department of Transportation, July 1999.
- [35] C. Cao, F. Lin, and Z. Lin, “Why event observation: observability revisited,” *Discrete Event Dynamic Systems: Theory and Applications*, vol. 7, pp. 127–149, 1997.
- [36] R. Vidal, A. Chiuso, and S. Soatto, “Observability and identifiability of jump linear systems,” in *Proceedings of the IEEE Conference on Decision and Control*, (Las Vegas, NV), 2002.

- [37] A. Bemporad, G. Ferrari-Trecate, and M. Morari, “Observability and controllability of piecewise affine and hybrid systems,” *IEEE Transactions on Control*, vol. 45, no. 10, pp. 1864–1876, October 2000.
- [38] C. Cao, “Supervisory control of a class of hybrid dynamic systems,” in *Proceedings of the 36th Midwest Symposium on Circuits and Systems*, (Detroit, MI), pp. 967–970, August 1993.
- [39] M. Mohrenschiltdt, “Hybrid systems: solutions, stability, and control,” in *Proceedings of the American Control Conference*, (Chicago, IL), June 2000.
- [40] A. Bayen and C. Tomlin, “Nonlinear hybrid automaton model for aircraft landing,” SUDAAR 737, Dept. of Aeronautics and Astronautics, Stanford University, Stanford, CA, 2001.
- [41] L. Prandtl and O. Tietjens, *Applied Hydro- and Aeromechanics*. New York: Dover Publications (Eng. Societies Monographs), 1957 (1934).
- [42] I. Kroo, *Aircraft Design: Synthesis and Analysis*. Stanford, CA: Desktop Aeronautics, 2001. Pre-release Version 0.99, <http://adg.stanford.edu/aa241/AircraftDesign.html>.
- [43] J. Roskam and C.-T. Lan, *Airplane Aerodynamics and Performance*. Lawrence, Kansas: Design, Analysis, and Research Corporation, 1997.
- [44] A. Flaig and R. Hilbig, “High-lift design for large civil aircraft,” in *AGARD Conference Proceedings 515*, (France), October 1992.
- [45] S. Rogers, K. Roth, H. Cao, J. Slotnick, M. Whitlock, S. Nash, and M. Baker, “Computation of viscous flow for a Boeing 777 aircraft in landing configuration,” in *AIAA Conference Proceedings*, no. 2000-4221, October 1992.
- [46] L. Jenkinson, P. Simpkin, and D. Rhodes, *Civil Jet Aircraft Design*. Reston, VA: American Institute of Aeronautics and Astronautics, Inc., 1999. <http://www.bh.com/companions/aerodata>.
- [47] C. Hynes, August 2001. Conversations at NASA.
- [48] T. Lambregts, “Automatic flight control: Concepts and methods.” FAA National Resource Specialist, Advanced Controls, 1995.
- [49] J. Lygeros, C. Tomlin, and S. Sastry, “Multiobjective hybrid controller synthesis,” in *Hybrid and Real-Time Systems* (O. Maler, ed.), LNCS 1201, pp. 109–123, Grenoble: Springer Verlag, 1997.
- [50] X. Koutsoukos, P. Antsaklis, J. Stiver, and M. Lemmon, “Supervisory control of hybrid systems,” *Proceedings of the IEEE*, vol. 88, no. 7, pp. 1026–1049, 2000.

- [51] M. Oishi, C. Tomlin, V. Gopal, and D. Godbole, “Addressing multiobjective control: Safety and performance through constrained optimization,” in *Hybrid Systems: Computation and Control* (M. Di Benedetto and A. Sangiovanni-Vincentelli, eds.), LNCS 2034, pp. 459–472, Springer Verlag, March 2001.
- [52] A. Degani and E. Wiener, “On the design of flight-deck procedures,” tech. rep., NASA Contractor Report 177642, NASA Ames, Moffett Field, CA, 1994.

<b>REPORT DOCUMENTATION PAGE</b>			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE November 2003		3. REPORT TYPE AND DATES COVERED Technical Memorandum
4. TITLE AND SUBTITLE  Discrete Abstractions of Hybrid Systems: Verification of Safety and Application to User-Interface Design			5. FUNDING NUMBERS NASA Ames Research Center to San Jose State University Foundation, under NCC2-798, through human-automation theory subelement (RTOP 548-40-12), DARPA under MICA Program administered by SPAWAR under contract N66001-01-C-8080, DoD MURI program administered by ONR under N00014-00-1-06637, NSF Graduate Research Fellowship	
6. AUTHOR(S)  Meeko Oishi (Stanford University), Claire Tomlin (Stanford University), Asaf Degani (Ames Research Center)				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)  Ames Research Center Moffett Field, CA 94035-1000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Stanford University, Stanford, CA 94305 National Aeronautics and Space Administration Washington, DC 20546-0001			10. SPONSORING/MONITORING AGENCY REPORT NUMBER  NASA/TM-2003-212803	
11. SUPPLEMENTARY NOTES Point of Contact: Asaf Degani, Ames Research Center, MS 269-4, Moffett Field, CA 94035-1000 (650) 604-0013				
12a. DISTRIBUTION/AVAILABILITY STATEMENT  Unclassified — Unlimited Subject Category 64                      Distribution: Standard Availability: NASA CASI (301) 621-0390			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words)  Human interaction with complex hybrid systems involves the user, the automation's discrete mode logic, and the underlying continuous dynamics of the physical system. Often the user-interface of such systems displays a reduced set of information about the entire system. In safety-critical systems, how can we identify user-interface designs which do not have adequate information, or which may confuse the user? Here we describe a methodology, based on hybrid system analysis, to verify that a user-interface contains information necessary to safely complete a desired procedure or task. Verification within a hybrid framework allows us to account for the continuous dynamics underlying the simple, discrete representations displayed to the user. We provide two examples: a car traveling through a yellow light at an intersection and an aircraft autopilot in a landing/go-around maneuver. The examples demonstrate the general nature of this methodology, which is applicable to hybrid systems (not fully automated) which have operational constraints we can pose in terms of <i>safety</i> . This methodology differs from existing work in hybrid system verification in that we directly account for the user's interactions with the system.				
14. SUBJECT TERMS  interface verification, interface analysis, hybrid systems, nonlinear systems, control theory, reachability, human interaction, pilot display, interface design			15. NUMBER OF PAGES 65	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT	